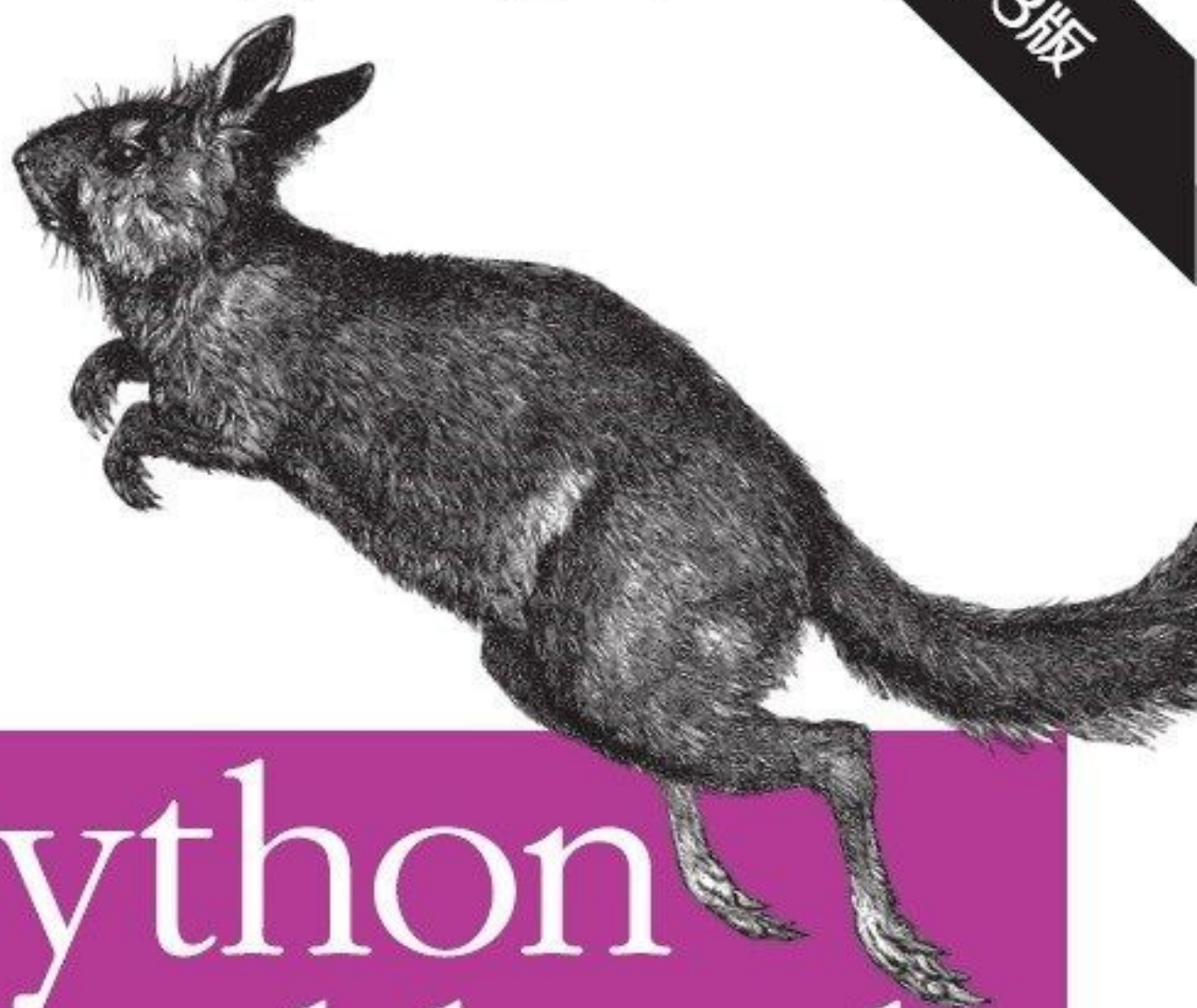


*Python Cookbook*  
*Recipes for Mastering Python 3*

第3版



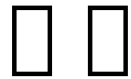
# Python Cookbook™

中文版

[美] David Beazley & Brian K. Jones 著  
陈舸 译

O'REILLY®

 人民邮电出版社  
POSTS & TELECOM PRESS



[□□□□](#)

[□□□□](#)

[□□□□](#)

[O'Reilly Media, Inc. □□](#)

[□□□□](#)

[□□](#)

[□1□ □□□□□□□□](#)

[1.1 □□□□□□□□□□□□](#)

[1.1.1 □□](#)

[1.1.2 □□□□](#)

[1.1.3 □□](#)

[1.2 □□□□□□□□□□□□□□□□](#)

[1.2.1 □□](#)

[1.2.2 □□□□](#)

[1.2.3 □□](#)

[1.3 □□□□N□□□□](#)

[1.3.1 □□](#)

[1.3.2 □□□□](#)

[1.3.3 □□](#)

[1.4 □□□□□□□□N□□□□](#)

[1.4.1 □□](#)

[1.4.2 □□□□](#)

[1.4.3 □□](#)

1.5 □□□□□□□

1.5.1 □□

1.5.2 □□□□

1.5.3 □□

1.6 □□□□□□□□□□□□□□

1.6.1 □□

1.6.2 □□□□

1.6.3 □□

1.7 □□□□□□□

1.7.1 □□

1.7.2 □□□□

1.7.3 □□

1.8 □□□□□□□□□□

1.8.1 □□

1.8.2 □□□□

1.8.3 □□

1.9 □□□□□□□□□□□□

1.9.1 □□

1.9.2 □□□□

1.9.3 □□

1.10 □□□□□□□□□□□□□□□□□□□□

1.10.1 □□

1.10.2 □□□□

1.10.3 □□

1.11 □□□□□

1.11.1 □□

1.11.2 □□□□

1.11.3 □□

1.12 □□□□□□□□□□□□□□

1.12.1 □□

1.12.2 □□□□

1.12.3 □□

1.13 □□□□□□□□□□□□

1.13.1 □□

1.13.2 □□□□

1.13.3 □□

1.14 □□□□□□□□□□□□□□

1.14.1 □□

1.14.2 □□□□

1.14.3 □□

1.15 □□□□□□□□□□

1.15.1 □□

1.15.2 □□□□

1.15.3 □□

1.16 □□□□□□□□

1.16.1 □□

1.16.2 □□□□

1.16.3 □□

1.17 □□□□□□□□

1.17.1 □□

1.17.2 □□□□

1.17.3 □□

1.18 □□□□□□□□□□□□

1.18.1 □□

1.18.2 □□□□

1.18.3 □□

1.19 □□□□□□□□□□□□

1.19.1 □□

1.19.2 □□□□

1.19.3 □□

1.20 □□□□□□□□□□□□

1.20.1 □□

1.20.2 □□□□

1.20.3 □□

□2□ □□□□□□

2.1 □□□□□□□□□□□□□□

2.1.1 □□

2.1.2 □□□□

2.1.3 □□

2.2 □□□□□□□□□□□□□□□□

2.2.1 □□

2.2.2 □□□□

2.2.3 □□

2.3 □□Shell□□□□□□□□□□

2.3.1 □□

2.3.2 □□□□

2.3.3 □□

2.4 □□□□□□□□□□□□

### 2.4.1

## 2.4.2

### 2.4.3 ☐ ☐

2.5

### 2.5.1 ☐☐

## 2.5.2 □□□□

### 2.5.3 ☐ ☐

## 2.6

## 2.6.1 ☐

## 2.6.2 □□□□

### 2.6.3 □□

## 2.7 □□□□□□□□□□□□□□□□

### 2.7.1 □□

### 2.7.2 関数

### 2.7.3 □□

## 2.8 □□□□□□□□□□□□□□

### 2.8.1 □□

## 2.8.2 関数

### 2.8.3 □□

## 2.9 Unicode

## 2.9.1 関数

## 2.9.2 四角形

### 2.9.3 閉曲面上のベクトル場

## 2.10 Unicode

### 2.10.1 □□

## 2.10.2 练习

[2.10.3 詳細](#)

[2.11 詳細](#)

[2.11.1 詳細](#)

[2.11.2 詳細](#)

[2.11.3 詳細](#)

[2.12 詳細](#)

[2.12.1 詳細](#)

[2.12.2 詳細](#)

[2.12.3 詳細](#)

[2.13 詳細](#)

[2.13.1 詳細](#)

[2.13.2 詳細](#)

[2.13.3 詳細](#)

[2.14 詳細](#)

[2.14.1 詳細](#)

[2.14.2 詳細](#)

[2.14.3 詳細](#)

[2.15 詳細](#)

[2.15.1 詳細](#)

[2.15.2 詳細](#)

[2.15.3 詳細](#)

[2.16 詳細](#)

[2.16.1 詳細](#)

[2.16.2 詳細](#)

[2.16.3 詳細](#)

[2.17 詳細HTML/XML](#)

2.17.1 □□

2.17.2 □□□□

2.17.3 □□

2.18 □□□□

2.18.1 □□

2.18.2 □□□□

2.18.3 □□

2.19 □□□□□□□□□□□□□□

2.19.1 □□

2.19.2 □□□□

2.19.3 □□

2.20 □□□□□□□□□□

2.20.1 □□

2.20.2 □□□□

2.20.3 □□

□3□ □□□□□□□□

3.1 □□□□□□□□

3.1.1 □□

3.1.2 □□□□

3.1.3 □□

3.2 □□□□□□□□□□

3.2.1 □□

3.2.2 □□□□

3.2.3 □□

3.3 □□□□□□□□□□

3.3.1 □□





3.10 □□□□□□□□□□

3.10.1 □□

3.10.2 □□□□

3.10.3 □□

3.11 □□□□

3.11.1 □□

3.11.2 □□□□

3.11.3 □□

3.12 □□□□

3.12.1 □□

3.12.2 □□□□

3.12.3 □□

3.13 □□□□5□□□

3.13.1 □□

3.13.2 □□□□

3.13.3 □□

3.14 □□□□□□□□□□

3.14.1 □□

3.14.2 □□□□

3.14.3 □□

3.15 □□□□□□□□□□

3.15.1 □□

3.15.2 □□□□

3.15.3 □□

3.16 □□□□□□□□□□□□

3.16.1 □□

3.16.2 □□□□

3.16.3 □□

□4□ □□□□□□□

4.1 □□□□□□□□□□

4.1.1 □□

4.1.2 □□□□

4.1.3 □□

4.2 □□□□

4.2.1 □□

4.2.3 □□□□

4.2.4 □□

4.3 □□□□□□□□□□

4.3.1 □□

4.3.2 □□□□

4.3.3 □□

4.4 □□□□□□

4.4.1 □□

4.4.2 □□□□

4.4.3 □□

4.5 □□□□

4.5.1 □□

4.5.2 □□□□

4.5.3 □□

4.6 □□□□□□□□□□

4.6.1 □□

4.6.2 □□□□

4.6.3 〇〇

4.7 〇〇〇〇〇〇〇〇〇〇

4.7.1 〇〇

4.7.2 〇〇〇〇

4.7.3 〇〇

4.8 〇〇〇〇〇〇〇〇〇〇〇〇〇〇〇〇

4.8.1 〇〇

4.8.2 〇〇〇〇

4.8.3 〇〇

4.9 〇〇〇〇〇〇〇〇〇〇〇〇〇〇

4.9.1 〇〇

4.9.2 〇〇〇〇

4.9.3 〇〇

4.10 〇〇〇-〇〇〇〇〇〇〇〇〇〇

4.10.1 〇〇

4.10.2 〇〇〇〇

4.10.3 〇〇

4.11 〇〇〇〇〇〇〇〇〇〇

4.11.1 〇〇

4.11.2 〇〇〇〇

4.11.3 〇〇

4.12 〇〇〇〇〇〇〇〇〇〇〇〇

4.12.1 〇〇

4.12.2 〇〇〇〇

4.12.3 〇〇

4.13 〇〇〇〇〇〇〇〇〇〇

[4.13.1](#) [□□](#)

[4.13.2](#) [□□□□](#)

[4.13.3](#) [□□](#)

[4.14](#) [□□□□□□□□□□](#)

[4.14.1](#) [□□](#)

[4.14.2](#) [□□□□](#)

[4.14.3](#) [□□](#)

[4.15](#) [□□□□□□□□□□□□□□□□□□□□](#)

[4.15.1](#) [□□](#)

[4.15.2](#) [□□□□](#)

[4.15.3](#) [□□](#)

[4.16](#) [□□□□□□while□□](#)

[4.16.1](#) [□□](#)

[4.16.2](#) [□□□□](#)

[4.16.3](#) [□□](#)

[□5□](#) [□□□I/O](#)

[5.1](#) [□□□□□□](#)

[5.1.1](#) [□□](#)

[5.1.2](#) [□□□□](#)

[5.1.3](#) [□□](#)

[5.2](#) [□□□□□□□□□□](#)

[5.2.1](#) [□□](#)

[5.2.2](#) [□□□□](#)

[5.2.3](#) [□□](#)

[5.3](#) [□□□□□□□□□□□□□□□□](#)

[5.3.1](#) [□□](#)

[5.3.2 関数](#)

[5.3.3 関数](#)

[5.4 関数型](#)

[5.4.1 関数](#)

[5.4.2 関数](#)

[5.4.3 関数](#)

[5.5 関数型と関数型](#)

[5.5.1 関数](#)

[5.5.2 関数](#)

[5.5.3 関数](#)

[5.6 関数型とIO関数](#)

[5.6.1 関数](#)

[5.6.2 関数](#)

[5.6.3 関数](#)

[5.7 関数型と関数型](#)

[5.7.1 関数](#)

[5.7.2 関数](#)

[5.7.3 関数](#)

[5.8 関数型と関数型](#)

[5.8.1 関数](#)

[5.8.2 関数](#)

[5.8.3 関数](#)

[5.9 関数型と関数型](#)

[5.9.1 関数](#)

[5.9.2 関数](#)

[5.9.3 関数](#)

5.10 □□□□□□□□□□

5.10.1 □□

5.10.2 □□□□

5.10.3 □□

5.11 □□□□□

5.11.1 □□

5.11.2 □□□□

5.11.3 □□

5.12 □□□□□□□□

5.12.1 □□

5.12.2 □□□□

5.12.3 □□

5.13 □□□□□□□□□

5.13.1 □□

5.13.2 □□□□

5.13.3 □□

5.14 □□□□□□□

5.14.1 □□

5.14.2 □□□□

5.14.3 □□

5.15 □□□□□□□□□□

5.15.1 □□

5.15.2 □□□□

5.15.3 □□

5.16 □□□□□□□□□□□□□□□□

5.16.1 □□

[5.16.2 関数](#)

[5.16.3 辞書](#)

[5.17 辞書とリストの操作](#)

[5.17.1 辞書](#)

[5.17.2 リスト](#)

[5.17.3 辞書](#)

[5.18 辞書とリストの操作](#)

[5.18.1 辞書](#)

[5.18.2 リスト](#)

[5.18.3 辞書](#)

[5.19 辞書とリストの操作](#)

[5.19.1 辞書](#)

[5.19.2 リスト](#)

[5.19.3 辞書](#)

[5.20 辞書とリストの操作](#)

[5.20.1 辞書](#)

[5.20.2 リスト](#)

[5.20.3 辞書](#)

[5.21 Pythonのインストール](#)

[5.21.1 辞書](#)

[5.21.2 リスト](#)

[5.21.3 辞書](#)

[6 辞書とリストの操作](#)

[6.1 CSVファイル](#)

[6.1.1 辞書](#)

[6.1.2 リスト](#)



[6.1.3 関数](#)

[6.2 関数JSON関数](#)

[6.2.1 関数](#)

[6.2.2 関数関数](#)

[6.2.3 関数](#)

[6.3 関数関数XML関数](#)

[6.3.1 関数](#)

[6.3.2 関数関数](#)

[6.3.3 関数](#)

[6.4 関数関数関数関数XML関数](#)

[6.4.1 関数](#)

[6.4.2 関数関数](#)

[6.4.3 関数](#)

[6.5 関数関数関数XML](#)

[6.5.1 関数](#)

[6.5.2 関数関数](#)

[6.5.3 関数](#)

[6.6 関数関数関数関数XML](#)

[6.6.1 関数](#)

[6.6.2 関数関数](#)

[6.6.3 関数](#)

[6.7 関数関数関数関数XML関数](#)

[6.7.1 関数](#)

[6.7.2 関数関数](#)

[6.7.3 関数](#)

[6.8 関数関数関数関数関数関数](#)

[6.8.1 □□](#)

[6.8.2 □□□□](#)

[6.8.3 □□](#)

[6.9 □□□□□□□□□□](#)

[6.9.1 □□](#)

[6.9.2 □□□□](#)

[6.9.3 □□](#)

[6.10 Base64□□□□□](#)

[6.10.1 □□](#)

[6.10.2 □□□□](#)

[6.10.3 □□](#)

[6.11 □□□□□□□□□□](#)

[6.11.1 □□](#)

[6.11.2 □□□□](#)

[6.11.3 □□](#)

[6.12 □□□□□□□□□□□□□□□□](#)

[6.12.1 □□](#)

[6.12.2 □□□□](#)

[6.12.3 □□](#)

[6.13 □□□□□□□](#)

[6.13.1 □□](#)

[6.13.2 □□□□](#)

[6.13.3 □□](#)

[□7□□□](#)

[7.1 □□□□□□□□□□□□□□□□](#)

[7.1.1 □□](#)

7.1.2 □□□□

7.1.3 □□

7.2 □□□□□□□□□□□□□□

7.2.1 □□

7.2.2 □□□□

7.2.3 □□

7.3 □□□□□□□□□□□□□□

7.3.1 □□

7.3.2 □□□□

7.3.3 □□

7.4 □□□□□□□□□□

7.4.1 □□

7.4.2 □□□□

7.4.3 □□

7.5 □□□□□□□□□□□□

7.5.1 □□

7.5.2 □□□□

7.5.3 □□

7.6 □□□□□□□□□□

7.6.1 □□

7.6.2 □□□□

7.6.3 □□

7.7 □□□□□□□□□□□□

7.7.1 □□

7.7.2 □□□□

7.7.3 □□

## 7.8 □□□N□□□□□□□□□□□□□□□□□□

### 7.8.1 □□

### 7.8.2 □□□□

### 7.8.3 □□

## 7.9 □□□□□□□□□□□□□□

### 7.9.1 □□

### 7.9.2 □□□□

### 7.9.3 □□

## 7.10 □□□□□□□□□□□□□□

### 7.10.1 □□

### 7.10.2 □□□□

### 7.10.3 □□

## 7.11 □□□□□□

### 7.11.1 □□

### 7.11.2 □□□□

### 7.11.3 □□

## 7.12 □□□□□□□□□□□□

### 7.12.1 □□

### 7.12.2 □□□□

### 7.12.3 □□

## □8□ □□□□

## 8.1 □□□□□□□□□□

### 8.1.1 □□

### 8.1.2 □□□□

### 8.1.3 □□

## 8.2 □□□□□□□□□□□□

8.2.1 □□

8.2.2 □□□□

8.2.3 □□

8.3 □□□□□□□□□□□□

8.3.1 □□

8.3.2 □□□□

8.3.3 □□

8.4 □□□□□□□□□□□□□□

8.4.1 □□

8.4.2 □□□□

8.4.3 □□

8.5 □□□□□□□□

8.5.1 □□

8.5.2 □□□□

8.5.3 □□

8.6 □□□□□□□□

8.6.1 □□

8.6.2 □□□□

8.6.3 □□

8.7 □□□□□□□□

8.7.1 □□

8.7.2 □□□□

8.7.3 □□

8.8 □□□□□□□□

8.8.1 □□

8.8.2 □□□□

8.8.3 □□

8.9 □□□□□□□□□□□□□□□□

8.9.1 □□

8.9.2 □□□□

8.9.3 □□

8.10 □□□□□□□□□□□□

8.10.1 □□

8.10.2 □□□□

8.10.3 □□

8.11 □□□□□□□□□□□□

8.11.1 □□

8.11.2 □□□□

8.11.3 □□

8.12 □□□□□□□□□□□□

8.12.1 □□

8.12.2 □□□□

8.12.3 □□

8.13 □□□□□□□□□□□□□□

8.13.1 □□

8.13.2 □□□□

8.13.3 □□

8.14 □□□□□□□□

8.14.1 □□

8.14.2 □□□□

8.14.3 □□

8.15 □□□□□□□□

[8.15.1](#) [□□](#)

[8.15.2](#) [□□□□](#)

[8.15.3](#) [□□](#)

[8.16](#) [□□□□□□□□□□](#)

[8.16.1](#) [□□](#)

[8.16.2](#) [□□□□](#)

[8.16.3](#) [□□](#)

[8.17](#) [□□□□□init□□□□□](#)

[8.17.1](#) [□□](#)

[8.17.2](#) [□□□□](#)

[8.17.3](#) [□□](#)

[8.18](#) [□Mixin□□□□□□□□](#)

[8.18.1](#) [□□](#)

[8.18.2](#) [□□□□](#)

[8.18.3](#) [□□](#)

[8.19](#) [□□□□□□□□□□□□□□](#)

[8.19.1](#) [□□](#)

[8.19.2](#) [□□□□](#)

[8.19.3](#) [□□](#)

[8.20](#) [□□□□□□□□□□□□□□□□□□□□](#)

[8.20.1](#) [□□](#)

[8.20.2](#) [□□□□](#)

[8.20.3](#) [□□](#)

[8.21](#) [□□□□□□□□](#)

[8.21.1](#) [□□](#)

[8.21.2](#) [□□□□](#)

8.21.3 □□

8.22 □□□□□□□□□□

8.22.1 □□

8.22.2 □□□□

8.22.3 □□

8.23 □□□□□□□□□□

8.23.1 □□

8.23.2 □□□□

8.23.3 □□

8.24 □□□□□□□□

8.24.1 □□

8.24.2 □□□□

8.24.3 □□

8.25 □□□□□□

8.25.1 □□

8.25.2 □□□□

8.25.3 □□

□9□ □□□

9.1 □□□□□□□□

9.1.1 □□

9.1.2 □□□□

9.1.3 □□

9.2 □□□□□□□□□□□□□□

9.2.1 □□

9.2.2 □□□□

9.2.3 □□



9.3 □□□□□□□□□□

9.3.1 □□

9.3.2 □□□□

9.3.3 □□

9.4 □□□□□□□□□□□□□□

9.4.1 □□

9.4.2 □□□□

9.4.3 □□

9.5 □□□□□□□□□□□□□□□□

9.5.1 □□

9.5.2 □□□□

9.5.3 □□

9.6 □□□□□□□□□□□□□□□□

9.6.1 □□

9.6.2 □□□□

9.6.3 □□

9.7 □□□□□□□□□□□□□□□□□□

9.7.1 □□

9.7.2 □□□□

9.7.3 □□

9.8 □□□□□□□□

9.8.1 □□

9.8.2 □□□□

9.8.3 □□

9.9 □□□□□□□□

9.9.1 □□

[9.9.2 □□□□](#)

[9.9.3 □□](#)

[9.10 □□□□□□□□□□□□□□](#)

[9.10.1 □□](#)

[9.10.2 □□□□](#)

[9.10.3 □□](#)

[9.11 □□□□□□□□□□□□□□□□](#)

[9.11.1 □□](#)

[9.11.2 □□□□](#)

[9.11.3 □□](#)

[9.12 □□□□□□□□□□□□](#)

[9.12.1 □□](#)

[9.12.2 □□□□](#)

[9.12.3 □□](#)

[9.13 □□□□□□□□□□□□](#)

[9.13.1 □□](#)

[9.13.2 □□□□](#)

[9.13.3 □□](#)

[9.14 □□□□□□□□□□](#)

[9.14.1 □□](#)

[9.14.2 □□□□](#)

[9.14.3 □□](#)

[9.15 □□□□□□□□□□□□□□](#)

[9.15.1 □□](#)

[9.15.2 □□□□](#)

[9.15.3 □□](#)

[9.16 `\*args` `\*\*kwargs`](#)

[9.16.1](#)

[9.16.2](#)

[9.16.3](#)

[9.17](#)

[9.17.1](#)

[9.17.2](#)

[9.17.3](#)

[9.18](#)

[9.18.1](#)

[9.18.2](#)

[9.18.3](#)

[9.19](#)

[9.19.1](#)

[9.19.2](#)

[9.19.3](#)

[9.20](#)

[9.20.1](#)

[9.20.2](#)

[9.20.3](#)

[9.21](#)

[9.21.1](#)

[9.21.2](#)

[9.21.3](#)

[9.22](#)

[9.22.1](#)

[9.22.2 関数](#)

[9.22.3 関数](#)

[9.23 関数型オブジェクト](#)

[9.23.1 関数](#)

[9.23.2 関数](#)

[9.23.3 関数](#)

[9.24 関数型オブジェクトPython](#)

[9.24.1 関数](#)

[9.24.2 関数](#)

[9.24.3 関数](#)

[9.25 Python関数型オブジェクト](#)

[9.25.1 関数](#)

[9.25.2 関数](#)

[9.25.3 関数](#)

[10 関数型オブジェクト](#)

[10.1 関数型オブジェクト](#)

[10.1.1 関数](#)

[10.1.2 関数](#)

[10.1.3 関数](#)

[10.2 関数型オブジェクト](#)

[10.2.1 関数](#)

[10.2.2 関数](#)

[10.2.3 関数](#)

[10.3 関数型オブジェクト](#)

[10.3.1 関数](#)

[10.3.2 関数](#)

[10.3.3](#) [□□](#)

[10.4](#) [□□□□□□□□□□](#)

[10.4.1](#) [□□](#)

[10.4.2](#) [□□□□](#)

[10.4.3](#) [□□](#)

[10.5](#) [□□□□□□□□□□□□□□□□□□□□](#)

[10.5.1](#) [□□](#)

[10.5.2](#) [□□□□](#)

[10.5.3](#) [□□](#)

[10.6](#) [□□□□□□](#)

[10.6.1](#) [□□](#)

[10.6.2](#) [□□□□](#)

[10.6.3](#) [□□](#)

[10.7](#) [□□□□zip□□□□□□□□□□](#)

[10.7.1](#) [□□](#)

[10.7.2](#) [□□□□](#)

[10.7.3](#) [□□](#)

[10.8](#) [□□□□□□□□□□](#)

[10.8.1](#) [□□](#)

[10.8.2](#) [□□□□](#)

[10.8.3](#) [□□](#)

[10.9](#) [□□□□□sys.path□](#)

[10.9.1](#) [□□](#)

[10.9.2](#) [□□□□](#)

[10.9.3](#) [□□](#)

[10.10](#) [□□□□□□□□□□□□□□□□](#)

10.10.1 □□

10.10.2 □□□□

10.10.3 □□

10.11 □□import□□□□□□□□□□□□

10.11.1 □□

10.11.2 □□□□

10.11.3 □□

10.12 □□□□□□□□□□

10.12.1 □□

10.12.2 □□□□

10.12.3 □□

10.13 □□□□□□□□□□

10.13.1 □□

10.13.2 □□□□

10.13.3 □□

10.14 □□□□Python□□

10.14.1 □□

10.14.2 □□□□

10.14.3 □□

10.15 □□□□□□□□

10.15.1 □□

10.15.2 □□□□

10.15.3 □□

□11□ □□□Web□□

11.1 □□□□□□□□HTTP□□□□

11.1.1 □□

[11.1.2 関数](#)

[11.1.3 関数](#)

[11.2 関数とTCP/IP](#)

[11.2.1 関数](#)

[11.2.2 関数](#)

[11.2.3 関数](#)

[11.3 関数とUDP/IP](#)

[11.3.1 関数](#)

[11.3.2 関数](#)

[11.3.3 関数](#)

[11.4 CIDRとIPv6](#)

[11.4.1 関数](#)

[11.4.2 関数](#)

[11.4.3 関数](#)

[11.5 関数とREST API](#)

[11.5.1 関数](#)

[11.5.2 関数](#)

[11.5.3 関数](#)

[11.6 XML-RPCとJSON-RPC](#)

[11.6.1 関数](#)

[11.6.2 関数](#)

[11.6.3 関数](#)

[11.7 関数とデータベース](#)

[11.7.1 関数](#)

[11.7.2 関数](#)

[11.7.3 関数](#)

11.8 [データベース](#)

11.8.1 [DB](#)

11.8.2 [DBMS](#)

11.8.3 [DB](#)

11.9 [データベースの設計](#)

11.9.1 [DB](#)

11.9.2 [DBMS](#)

11.9.3 [DB](#)

11.10 [データベースのセキュリティ](#)

11.10.1 [DB](#)

11.10.2 [DBMS](#)

11.10.3 [DB](#)

11.11 [データベースのネットワーク](#)

11.11.1 [DB](#)

11.11.2 [DBMS](#)

11.11.3 [DB](#)

11.12 [データベースのI/O](#)

11.12.1 [DB](#)

11.12.2 [DBMS](#)

11.12.3 [DB](#)

11.13 [データベースの最適化](#)

11.13.1 [DB](#)

11.13.2 [DBMS](#)

11.13.3 [DB](#)

12 [まとめ](#)

12.1 [データベース](#)



12.1.1 □□

12.1.2 □□□□

12.1.3 □□

12.2 □□□□□□□□□□

12.2.1 □□

12.2.2 □□□□

12.2.3 □□

12.3 □□□□□

12.3.1 □□

12.3.2 □□□□

12.3.3 □□

12.4 □□□□□□

12.4.1 □□

12.4.2 □□□□

12.4.3 □□

12.5 □□□□

12.5.1 □□

12.5.2 □□□□

12.5.3 □□

12.6 □□□□□□□□

12.6.1 □□

12.6.2 □□□□

12.6.3 □□

12.7 □□□□□

12.7.1 □□

12.7.2 □□□□

[12.7.3](#) [□□](#)

[12.8](#) [□□□□□□□□](#)

[12.8.1](#) [□□](#)

[12.8.2](#) [□□□□](#)

[12.8.3](#) [□□](#)

[12.9](#) [□□□□GIL□□□□□](#)

[12.9.1](#) [□□](#)

[12.9.2](#) [□□□□](#)

[12.9.3](#) [□□](#)

[12.10](#) [□□□□Actor□□](#)

[12.10.1](#) [□□](#)

[12.10.2](#) [□□□□](#)

[12.10.3](#) [□□](#)

[12.11](#) [□□□□□/□□□□□□□](#)

[12.11.1](#) [□□](#)

[12.11.2](#) [□□□□](#)

[12.11.3](#) [□□](#)

[12.12](#) [□□□□□□□□□□□□□□](#)

[12.12.1](#) [□□](#)

[12.12.2](#) [□□□□](#)

[12.12.3](#) [□□](#)

[12.13](#) [□□□□□□□□](#)

[12.13.1](#) [□□](#)

[12.13.2](#) [□□□□](#)

[12.13.3](#) [□□](#)

[12.14](#) [□UNIX□□□□□□□](#)

12.14.1 □□

12.14.2 □□□□

12.14.3 □□

□13□ □□□□□□□□□□

13.1 □□□□□□□□□□□□□□□□□□□□□□□□

13.1.1 □□

13.1.2 □□□□

13.1.3 □□

13.2 □□□□□□□□□□□□

13.2.1 □□

13.2.2 □□□□

13.2.3 □□

13.3 □□□□□□□□

13.3.1 □□

13.3.2 □□□□

13.3.3 □□

13.4 □□□□□□□□□□□□□□

13.4.1 □□

13.4.2 □□□□

13.4.3 □□

13.5 □□□□□□□□

13.5.1 □□

13.5.2 □□□□

13.5.3 □□

13.6 □□□□□□□□□□□□□□

13.6.1 □□

13.6.2 □□□□

13.6.3 □□

13.7 □□□□□□□□□□

13.7.1 □□

13.7.2 □□□□

13.7.3 □□

13.8 □□□□□□□□□□

13.8.1 □□

13.8.2 □□□□

13.8.3 □□

13.9 □□□□□□□□□□

13.9.1 □□

13.9.2 □□□□

13.9.3 □□

13.10 □□□□□□□□

13.10.1 □□

13.10.2 □□□□

13.10.3 □□

13.11 □□□□□□□□□□

13.11.1 □□

13.11.2 □□□□

13.11.3 □□

13.12 □□□□□□□□□□

13.12.1 □□

13.12.2 □□□□

13.12.3 □□

[13.13 □□□□□□□□](#)

[13.13.1 □□](#)

[13.13.2 □□□□](#)

[13.13.3 □□](#)

[13.14 □□□□CPU□□□□□□□](#)

[13.14.1 □□](#)

[13.14.2 □□□□](#)

[13.14.3 □□](#)

[13.15 □□Web□□□](#)

[13.15.1 □□](#)

[13.15.2 □□□□](#)

[13.15.3 □□](#)

[14 □□□□□□□□□□](#)

[14.1 □□□□□stdout□□□□](#)

[14.1.1 □□](#)

[14.1.2 □□□□](#)

[14.1.3 □□](#)

[14.2 □□□□□□□□□□□□](#)

[14.2.1 □□](#)

[14.2.2 □□□□](#)

[14.2.3 □□](#)

[14.3 □□□□□□□□□□□□](#)

[14.3.1 □□](#)

[14.3.2 □□□□](#)

[14.3.3 □□](#)

[14.4 □□□□□□□□□□□□□□□□](#)

14.4.1 □□

14.4.2 □□□□

14.4.3 □□

14.5 □□□□□□□□□□□□□□□□

14.5.1 □□

14.5.2 □□□□

14.5.3 □□

14.6 □□□□□□

14.6.1 □□

14.6.2 □□□□

14.6.3 □□

14.7 □□□□□□□□

14.7.1 □□

14.7.2 □□□□

14.7.3 □□

14.8 □□□□□□□□

14.8.1 □□

14.8.2 □□□□

14.8.3 □□

14.9 □□□□□□□□□□□□□□

14.9.1 □□

14.9.2 □□□□

14.9.3 □□

14.10 □□□□□□□□□□

14.10.1 □□

14.10.2 □□□□

14.10.3 □□

14.11 □□□□□□

14.11.1 □□

14.11.2 □□□□

14.11.3 □□

14.12 □□□□□□□□□□□□□□

14.12.1 □□

14.12.2 □□□□

14.12.3 □□

14.13 □□□□□□□□□□□□□□

14.13.1 □□

14.13.2 □□□□

14.13.3 □□

14.14 □□□□□□□□□□

14.14.1 □□

14.14.2 □□□□

14.14.3 □□

□15□ C□□□□

15.1 □□ctypes□□□C□□

15.1.1 □□

15.1.2 □□□□

15.1.3 □□

15.2 □□□□□C□□□□□□□

15.2.1 □□

15.2.2 □□□□

15.2.3 □□

## 15.3 [Python 2.7 64-bit 32-bit](#)

### 15.3.1 [Python 2.7 64-bit](#)

### 15.3.2 [Python 2.7 32-bit](#)

### 15.3.3 [Python 2.7 64-bit](#)

## 15.4 [Python 3.5 64-bit 32-bit](#)

### 15.4.1 [Python 3.5 64-bit](#)

### 15.4.2 [Python 3.5 32-bit](#)

### 15.4.3 [Python 3.5 64-bit](#)

## 15.5 [Python 3.5 64-bit 32-bit C API](#)

### 15.5.1 [Python 3.5 64-bit](#)

### 15.5.2 [Python 3.5 32-bit](#)

### 15.5.3 [Python 3.5 64-bit](#)

## 15.6 [Python 3.5 64-bit 32-bit Python](#)

### 15.6.1 [Python 3.5 64-bit](#)

### 15.6.2 [Python 3.5 32-bit](#)

### 15.6.3 [Python 3.5 64-bit](#)

## 15.7 [Python 3.5 64-bit 32-bit GIL](#)

### 15.7.1 [Python 3.5 64-bit](#)

### 15.7.2 [Python 3.5 32-bit](#)

### 15.7.3 [Python 3.5 64-bit](#)

## 15.8 [Python 3.5 64-bit 32-bit Python C API](#)

### 15.8.1 [Python 3.5 64-bit](#)

### 15.8.2 [Python 3.5 32-bit](#)

### 15.8.3 [Python 3.5 64-bit](#)

## 15.9 [Python 3.5 64-bit 32-bit Swig](#)

### 15.9.1 [Python 3.5 64-bit](#)



[15.9.2 関数](#)

[15.9.3 関数](#)

[15.10 CythonによるC](#)

[15.10.1 関数](#)

[15.10.2 関数](#)

[15.10.3 関数](#)

[15.11 CythonによるC](#)

[15.11.1 関数](#)

[15.11.2 関数](#)

[15.11.3 関数](#)

[15.12 関数によるC](#)

[15.12.1 関数](#)

[15.12.2 関数](#)

[15.12.3 関数](#)

[15.13 関数NULLによるC](#)

[15.13.1 関数](#)

[15.13.2 関数](#)

[15.13.3 関数](#)

[15.14 UnicodeによるC](#)

[15.14.1 関数](#)

[15.14.2 関数](#)

[15.14.3 関数](#)

[15.15 CによるPython](#)

[15.15.1 関数](#)

[15.15.2 関数](#)

[15.15.3 関数](#)

15.16 □□□□□□□□C□□□□□□

15.16.1 □□

15.16.2 □□□□

15.16.3 □□

15.17 □□□□□□C□□□□

15.17.1 □□

15.17.2 □□□□

15.17.3 □□

15.18 □□□□□□□□C□□□□

15.18.1 □□

15.18.2 □□□□

15.18.3 □□

15.19 □C□□□□□□□□

15.19.1 □□

15.19.2 □□□□

15.19.3 □□

15.20 □C□□□□□□□□

15.20.1 □□

15.20.2 □□□□

15.20.3 □□

15.21 □□□□□

15.21.1 □□

15.21.2 □□□□

15.21.3 □□

□□A □□□□

A.1 □□□□

A.2 Python

A.3

□□□□

□□□Python Cookbook□□3□□□□□

ISBN□978-7-115-37959-7

□□□□□□□□□□□□□□□□□□□□□□□□

---

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□

---

• □ [□] David Beazley Brian  
K.Jones

□ □ □

□□□□ □□□

• □□□□□□□□□□□ □□□□□□□□□□11□

□□ 100164 □□□□  
315@ptpress.com.cn

□□ <http://www.ptpress.com.cn>

- □□□□□□(010)81055410

□□□□□□(010)81055315

□□□□

Copyright © 2013 by O'Reilly  
Media, Inc.

Simplified Chinese Edition, jointly  
published by O'Reilly Media, Inc. and Posts  
& Telecom Press, 2015. Authorized  
translation of the English edition, 2011  
O'Reilly Media, Inc., the owner of all rights  
to publish and sell the same.

All rights reserved including the rights  
of reproduction in whole or in part in any  
form.

□□□□□□□□O'Reilly Media, Inc.□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□

□□□□□□□□□□

□□□□

□□□□Python□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□I/O□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□Web□□□□□□□□□□□□□□□□□□□□□□□□□□□□C□□□□□  
□□

□□□□Python□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□Python 3.3□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□Python□□□□□□□□□□

# O'Reilly Media, Inc.

O'Reilly Media  
 1978 O'Reilly  
 —“”  
 O'Reilly

O'Reilly " " GNN Make DIY O'Reilly O'Reilly O'Reilly — —

□ □ □ □

# "O'Reilly Radar"

—Wired

"O'Reilly [redacted]  
[redacted]"



## —Business 2.0

“O’Reilly Conference”

—CRN

“O’Reilly”

—Irish Times

“Tim \_\_\_\_\_  
 \_\_\_\_\_Yogi Berra \_\_\_\_\_‘ \_\_\_\_\_  
 \_\_\_\_\_’ \_\_\_\_\_Tim \_\_\_\_\_  
 \_\_\_\_\_”

—Linux Journal

□□□□

**David Beazley** □□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□  
Python□□□□□□□□□□□□□□□□□□□□□□Swig□  
PLY□□□□□□□□□□□□Python Essential  
Reference□□□□□□□C□C++□□□□□□□□□□□□□□  
□□□□□□

**Brain K. Jones** □□□□□□□□□□□□□□□□  
□□

□□

□2008□□□□□□□□□□Python□□□□□□□  
Python 3□□□□□□□□□□□□□□Python 3□□□□□□  
□□□□□□□□□□□□□2013□□□□□□Python□□□□□  
□□□□□□□□□□Python 2□□□Python 3□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□Python 3□□□□□□  
□□□□□□

□□□Python 3□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□Python 3□□□□  
□□□□□□□Python 3.3□□□□□□□□□□□□□□□□  
Python□□□□“□□”□□□□□□□□□□□□□□□□□□□  
Python 3.3□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□Python 3□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□Python 3□□□□□□□□□□□□□□□□□□  
□□□□□□□Python 3□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□Python□□□□□□□□ActiveState□  
Python□□□□Stack Overflow□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□Python 2□□□□□□□□□□□□□□  
Python□□□□□□□2.3□□□2.4□□□□□□□□□□□□□□

Python 3.3 Python 3

```

Python 3
Python

```

Python

Python 3

Python

Python

Python

Python

C

Python

Python

Python



--	--	--	--	--	--

<http://github.com/dabeaz/python-cookbook>
[bug](#)

□ □ □ □ □ □

A 7x25 grid of squares. The text "O'Reilly" is centered in the middle row (row 4), with the "O" starting at column 15 and the "y" ending at column 23. The text is in a black, sans-serif font. The grid is composed of 175 squares in total, arranged in 7 rows and 25 columns.

Python Cookbook, 3rd edition, by David Beazley and Brian K. Jones (O'Reilly) Copyright 2013 David Beazley and Brian Jones, 978-1-449-34037-7

permissions@oreilly.com

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□

□□□

O'Reilly Media Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

□□□

□□□□□□□□□□□□2□□□□□C□807□  
□100035□

□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□

[http://oreil.ly/python\\_cookbook\\_3e](http://oreil.ly/python_cookbook_3e) □□□□□□

□□□□□□□□□□□□□□□□□□□□

bookquestions@oreilly.com

http://www.oreilly.com

Facebook: <http://facebook.com/oreilly>

Twitter: <http://twitter.com/oreillymedia>

YouTube:  
<http://www.youtube.com/oreillymedia>

11

Jake Vanderplas  
 Robert Kern  
 Andrea Crotti  
 Python  
 2  
 Alex Martelli  
 Anna Ravenscroft  
 David  
 Ascher  
 3

# David Beazley

Paula 6 Python Ned Batchelder



Travis Oliphant □ Peter Wang □ Brain Van de  
Ven □ Hugo Shi □ Raymond Hettinger □  
Michael Foord □ Daniel Klein □ □ □ □ □ □ □ □ □ □  
□ O'Reilly □  
Meghan Blanchette □ Rachel Roumeliotis □  
□  
□ □ □ □ □ □ □ □ Python □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □  
□ □ □

# David M.Beazley

<http://www.dabeaz.com>

<https://twitter.com/dabeaz>

# Brain Jones

David Beazley  
O'Reilly  
Meghan Blanchette  
Rachel Roumeliotis  
Natasha  
Python  
Python

# Brain K.Jones

<http://www.protocolostomy.com>

<https://twitter.com/bkjones>

# 1 集合数据类型

Python 集合数据类型包括 `list`、`set`、`dictionary`、`collections` 等。其中 `list` 是有序集合，`set` 是无序集合，`dictionary` 是键值对集合，`collections` 是集合的扩展库。

## 1.1 集合的基本操作

### 1.1.1 集合的创建

集合的创建可以通过 `set()` 函数或花括号 `{}` 来实现。例如，创建一个包含元素 1、2、3 的集合：

### 1.1.2 集合的遍历

集合的遍历可以通过 `for` 循环来实现。例如，遍历集合 `s` 中的元素：

```
>>> p = (4, 5)
>>> x, y = p
>>> x
4
>>> y
5
```



```

>>> s = 'Hello'
>>> a, b, c, d, e = s
>>> a
'H'
>>> b
'e'
>>> e
'o'
>>>

```

Python

```

>>> data = [ 'ACME', 50, 91.1, (2012, 12, 21) ]
>>> _, shares, price, _ = data
>>> shares
50
>>> price
91.1
>>>

```

## 1.2

### 1.2.1

N

N “too many values to

unpack

## 1.2.2

Python

```
def
drop_first_last(grades):
    first, *middle, last = grades
    return
avg(middle)
```

```
>>> record = ('Dave', 'dave@example.com', '773-555-1212',
'847-555-1212')
>>> name, email, *phone_numbers = user_record
>>> name
'Dave'
>>> email
'dave@example.com'
>>> phone_numbers
['773-555-1212', '847-555-1212']
>>>
```

phone\_numbers = phone\_numbers + phone\_numbers

8

7

```
*trailing_qtrs, current_qtr = sales_record
trailing_avg = sum(trailing_qtrs) / len(trailing_qtrs)
return
avg_comparison(trailing_avg, current_qtr)
```

Python

```
>>> *trailing, current = [10, 8, 7, 1, 9, 5, 10, 3]
>>> trailing
[10, 8, 7, 1, 9, 5, 10]
>>> current
3
```

## 1.2.3

1 \*

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□

\*□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□

```
records = [  
    ('foo', 1, 2),  
    ('bar', 'hello'),  
    ('foo', 3, 4),  
]  
  
def  
  
do_foo(x, y):  
    print  
  
    ('foo', x, y)  
  
def  
  
do_bar(s):  
    print  
  
    ('bar', s)  
  
for  
  
tag, *args in  
  
records:  
    if  
  
    tag == 'foo':  
        do_foo(*args)  
    elif  
  
    tag == 'bar':  
        do_bar(*args)
```



---

字符串的splitting  
字符串\*字符串

```
>>> line = 'nobody:*:-2:-2:Unprivileged
User:/var/empty:/usr/bin/false'
>>> uname, *fields, homedir, sh = line.split(':')
>>> uname
'nobody'
>>> homedir
'/var/empty'
>>> sh
'/usr/bin/false'
>>>
```

字符串的splitting  
字符串\*字符串  
ign ignored

```
>>> record = ('ACME', 50, 123.45, (12, 18, 2012))
>>> name, *_ , (*_, year) = record
>>> name
'ACME'
>>> year
2012
>>>
```

\*字符串  
字符串

```
>>> items = [1, 10, 7, 4, 5, 9]
>>> head, *tail = items
>>> head
```

```
1
>>> tail
[10, 7, 4, 5, 9]
>>>
```

```
>>> def
sum(items):
...
head, *tail = items
...     return
head + sum(tail) if
tail else
head
...

>>> sum(items)
36
>>>
```

```

Python

```

### 1.3 1111N1111

## 1.3.1 deque

collections.deque는 리스트와 유사한 자료구조로, 양쪽 끝에서 요소를 추가하고 삭제할 수 있다. 리스트와 달리, deque는 O(1)의 시간 복잡도로 양쪽 끝에서 요소를 추가하고 삭제할 수 있다.

## 1.3.2 collections.deque

collections.deque는 리스트와 유사한 자료구조로, 양쪽 끝에서 요소를 추가하고 삭제할 수 있다. 리스트와 달리, deque는 O(1)의 시간 복잡도로 양쪽 끝에서 요소를 추가하고 삭제할 수 있다. N은 deque의 최대 길이이다.

```
from collections import deque

def search(lines, pattern, history=5):
    previous_lines = deque(maxlen=history)
    for line in lines:
        if pattern in line:
            yield line, previous_lines
        previous_lines.append(line)

# Example use on a file

if __name__ == '__main__':
    with open('somefile.txt') as f:
        for line, prevlines in search(f, 'python', 5):
            for pline in prevlines:
                print(pline, end='')
            print(line, end='')
            print('-'*20)
```

## 1.3.3 defaultdict

deque() 方法返回一个空的双端队列。deque() 方法还可以接受一个可迭代对象作为参数，将可迭代对象中的元素添加到双端队列中。deque() 方法还可以接受一个 maxlen 参数，指定双端队列的最大长度。如果 maxlen 参数被指定，那么双端队列的长度将不会超过 maxlen。deque() 方法还可以接受一个 maxlen 参数，指定双端队列的最大长度。如果 maxlen 参数被指定，那么双端队列的长度将不会超过 maxlen。4.3

deque(maxlen=N) 方法返回一个空的双端队列，其最大长度为 N。deque(maxlen=N) 方法还可以接受一个可迭代对象作为参数，将可迭代对象中的元素添加到双端队列中。deque(maxlen=N) 方法还可以接受一个 maxlen 参数，指定双端队列的最大长度。如果 maxlen 参数被指定，那么双端队列的长度将不会超过 maxlen。

```
>>> q = deque(maxlen=3)
>>> q.append(1)
>>> q.append(2)
>>> q.append(3)
>>> q
deque([1, 2, 3], maxlen=3)
>>> q.append(4)
>>> q
deque([2, 3, 4], maxlen=3)
>>> q.append(5)
>>> q
deque([3, 4, 5], maxlen=3)
```

deque 对象支持 append() 和 del 方法。append() 方法将元素添加到双端队列的右侧。del 方法删除双端队列中的元素。del 方法还可以接受一个索引参数，指定要删除的元素的位置。del 方法还可以接受一个切片参数，指定要删除的元素的范围。del 方法还可以接受一个 maxlen 参数，指定双端队列的最大长度。如果 maxlen 参数被指定，那么双端队列的长度将不会超过 maxlen。

deque 对象支持 deque() 方法。deque() 方法返回双端队列中的元素。deque() 方法还可以接受一个 maxlen 参数，指定双端队列的最大长度。如果 maxlen 参数被指定，那么双端队列的长度将不会超过 maxlen。

```
>>> q = deque()
>>> q.append(1)
>>> q.append(2)
>>> q.append(3)
>>> q
deque([1, 2, 3])
```

```
>>> q.appendleft(4)
>>> q
deque([4, 1, 2, 3])
>>> q.pop()
3
>>> q
deque([4, 1, 2])
>>> q.popleft()
4
```

$O(1)$   
 $O(N)$

## 1.4 □□□□□□□N□□□

### 1.4.1 〇〇

N

## 1.4.2 思考問題

```

heapq————nlargest()
nsmallest()————

```

```
import heapq

nums = [1, 8, 2, 23, 7, -4, 18, 23, 42, 37, 2]
print(heapq.nlargest(3, nums)) # Prints [42, 37, 23]

print(heapq.nsmallest(3, nums)) # Prints [-4, 1, 2]
```

key를 사용하여  
정렬한다

```
portfolio = [  
    {'name': 'IBM', 'shares': 100, 'price': 91.1},  
    {'name': 'AAPL', 'shares': 50, 'price': 543.22},  
    {'name': 'FB', 'shares': 200, 'price': 21.09},  
    {'name': 'HPQ', 'shares': 35, 'price': 31.75},  
    {'name': 'YHOO', 'shares': 45, 'price': 16.35},  
    {'name': 'ACME', 'shares': 75, 'price': 115.65}  
]  
  
cheap = heapq.nsmallest(3, portfolio, key=lambda  
s: s['price'])  
expensive = heapq.nlargest(3, portfolio, key=lambda  
s: s['price'])
```

## 1.4.3

N개의 원소를 가진 배열을 N개의 원소를 가진 배열로  
정렬한다

```
>>> nums = [1, 8, 2, 23, 7, -4, 18, 23, 42, 37, 2]  
>>> import heapq  
  
>>> heap = list(nums)
```

```
>>> heapq.heapify(heap)
>>> heap
[-4, 2, 1, 23, 7, 2, 18, 23, 42, 37, 8]
>>>
```

heapq.heap[0] 是堆的根节点，即堆顶元素。  
 heapq.heappop() 弹出堆顶元素，并返回该元素。  
 该操作的时间复杂度为  $O(\log N)$ ，其中  $N$  是堆的大小。  
 该操作的时间复杂度为  $O(\log N)$ ，其中  $N$  是堆的大小。

```
>>> heapq.heappop(heap)
-4
>>> heapq.heappop(heap)
1
>>> heapq.heappop(heap)
2
```

heapq.nlargest() 和 heapq.nsmallest() 用于返回堆中最大的  $n$  个元素和最小的  $n$  个元素。  
 当  $N=1$  时，min() 和 max() 分别返回堆中的最小值和最大值。  
 该操作的时间复杂度为  $O(N \log N)$ ，其中  $N$  是堆的大小。  
 该操作的时间复杂度为  $O(N \log N)$ ，其中  $N$  是堆的大小。  
 该操作的时间复杂度为  $O(N \log N)$ ，其中  $N$  是堆的大小。  
 该操作的时间复杂度为  $O(N \log N)$ ，其中  $N$  是堆的大小。

该操作的时间复杂度为  $O(N \log N)$ ，其中  $N$  是堆的大小。  
 该操作的时间复杂度为  $O(N \log N)$ ，其中  $N$  是堆的大小。

heapq

## 1.5

### 1.5.1

pop

### 1.5.2

heapq

```
import heapq
class PriorityQueue:

    def __init__(self):
        self._queue = []
        self._index = 0

    def push(self, item, priority):
        heapq.heappush(self._queue, (-priority, self._index,
item))
        self._index += 1

    def pop(self):
        return heapq.heappop(self._queue)[-1]
```



```

>>> class Item:
...     def __init__(self, name):
...         self.name = name
...     def __repr__(self):
...         return 'Item({!r})'.format(self.name)
...
>>> q = PriorityQueue()
>>> q.push(Item('foo'), 1)
>>> q.push(Item('bar'), 5)
>>> q.push(Item('spam'), 4)
>>> q.push(Item('grok'), 1)
>>> q.pop()
Item('bar')
>>> q.pop()
Item('spam')
>>> q.pop()
Item('foo')
>>> q.pop()
Item('grok')
>>>

```

pop()은 우선순위가 가장 낮은 원소를 반환합니다.  
 이 예에서는 'bar'가 가장 높은 우선순위를 가지므로 먼저 반환되며,  
 그 다음 'spam', 'foo', 그리고 'grok' 순서로 반환됩니다.

### 1.5.3 heapq

heapq 모듈은 heapq.heappush()과 heapq.heappop()을 사용하여  
 우선순위 큐를 구현합니다. heapq 모듈은 Python 2.5에서  
 처음 도입되었으며, Python 1.4에서는 heapq 모듈이 없었습니다.  
 heapq 모듈은 push()와 pop() 메서드를 제공합니다.

时间复杂度  $O(\log N)$  比  $N$  小得多，因此堆排序的时间复杂度是  $O(N \log N)$ 。

堆排序的算法步骤如下：  
1. 将待排序的数组 `arr` 构建成一个堆。  
2. 将堆顶元素（最大值）与堆尾元素交换。  
3. 将新的堆顶元素下沉，使其满足堆的性质。  
4. 重复步骤 2 和 3，直到堆中只剩下一个元素。

在实现堆排序时，我们通常使用一个辅助数组 `index` 来记录每个元素在堆中的位置。初始时，`index[i] = i`。当交换堆顶元素和堆尾元素后，我们需要更新 `index` 数组，并重新调整堆。

我们定义一个 `Item` 类，用于封装元素及其在堆中的位置。

```
>>> a = Item('foo')
>>> b = Item('bar')
>>> a < b
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unorderable types: Item() < Item()
>>>
```

为了解决这个问题，我们可以将 `Item` 类重写，使其支持比较操作。或者，我们可以将堆中的元素表示为元组 `(priority, item)`，这样可以直接比较元组的第一个元素（即优先级）。

```
>>> a = (1, Item('foo'))
>>> b = (5, Item('bar'))
>>> a < b
True
>>> c = (1, Item('grok'))
```

```
>>> a < c
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unorderable types: Item() < Item()
>>>
```

元组(priority, index, item)
 的第三个元素是Item对象。Python
 无法比较Item对象。

```
>>> a = (1, 0, Item('foo'))
>>> b = (5, 1, Item('bar'))
>>> c = (1, 2, Item('grok'))
>>> a < b
True
>>> a < c
True
>>>
```

在Python 2.3中，元组
 的第三个元素是字符串。

在Python 2.3中，元组
 的第三个元素是字符串。

## 1.6 堆排序

### 1.6.1 堆

```
{}[multidict]{}
```

## 1.6.2 □□□□

The diagram consists of three rows of blocks. The top row has 15 blocks. The middle row has 20 blocks. The bottom row has 7 blocks. The blocks are arranged in a way that suggests a subtraction problem: 20 minus 15 equals 5.

```
d = {
    'a' : [1, 2, 3],
    'b' : [4, 5]
}

e = {
    'a' : {1, 2, 3},
    'b' : {4, 5}
}
```

```
collections.defaultdict
collections.defaultdict
```

```
from collections import defaultdict

d = defaultdict(list)
d['a'].append(1)
d['a'].append(2)
```

```
d['b'].append(4)
...

d = defaultdict(set)
d['a'].add(1)
d['a'].add(2)
d['b'].add(4)
...
```

defaultdict 是字典的工厂函数，它返回一个字典，字典的默认值是 set。
 使用 defaultdict 可以简化代码，避免在字典中不存在键时抛出 KeyError。
 使用 defaultdict 的 setdefault() 方法可以返回字典中键对应的值，如果键不存在，则返回默认值并插入字典。

```
d = {} # A regular dictionary
d.setdefault('a', []).append(1)
d.setdefault('a', []).append(2)
d.setdefault('b', []).append(4)
...
```

defaultdict 的 setdefault() 方法返回字典中键对应的值，如果键不存在，则返回默认值并插入字典。
 使用 defaultdict 可以简化代码，避免在字典中不存在键时抛出 KeyError。

## 1.6.3 defaultdict

defaultdict 是字典的工厂函数，它返回一个字典，字典的默认值是 set。
 使用 defaultdict 可以简化代码，避免在字典中不存在键时抛出 KeyError。

```
d = {}
for
key, value in
```

```

pairs:
    if
key not in
d:
    d[key] = []
    d[key].append(value)

```

defaultdict

```

d = defaultdict(list)
for
key, value in
pairs:
    d[key].append(value)

```

1.15

## 1.7

### 1.7.1

## 1.7.2 OrderedDict

collections 모듈의 OrderedDict 클래스는 딕셔너리 객체에서 항목의 순서를 기억할 수 있는 딕셔너리이다.

```
from collections import OrderedDict

d = OrderedDict()
d['foo'] = 1
d['bar'] = 2
d['spam'] = 3
d['grok'] = 4

# Outputs "foo 1", "bar 2", "spam 3", "grok 4"

for key in d:
    print(key, d[key])
```

OrderedDict 객체는 JSON 데이터와 호환된다. OrderedDict 객체를 JSON 문자열로 변환하거나 JSON 문자열을 OrderedDict 객체로 변환할 수 있다.

```
>>> import json
>>> json.dumps(d)
'{"foo": 1, "bar": 2, "spam": 3, "grok": 4}'
>>>
```

## 1.7.3 defaultdict

OrderedDict 是字典的有序版本。它保持了插入的顺序，并且支持快速查找。它通常用于需要保持字典顺序的场景，例如在序列化数据时。

OrderedDict 是字典的有序版本。它保持了插入的顺序，并且支持快速查找。它通常用于需要保持字典顺序的场景，例如在序列化数据时。OrderedDict 与 CSV 文件的兼容性非常好，可以轻松地处理大量数据（例如 100000 行）。OrderedDict 与 OrderedDict 的兼容性非常好，可以轻松地处理大量数据（例如 100000 行）。

## 1.8 字典

### 1.8.1 字典

字典是 Python 中最常用的数据结构之一。它用于存储键值对，并且支持快速查找。字典的键必须是不可变的，而值可以是任意的 Python 对象。

### 1.8.2 字典

字典的键必须是不可变的，而值可以是任意的 Python 对象。

```
prices = {
    'ACME': 45.23,
    'AAPL': 612.78,
    'IBM': 205.55,
    'HPQ': 37.20,
    'FB': 10.75
}
```



```
}
```

zip() returns a zip object that is an iterator of tuples where each tuple is the key-value pair from the dictionary.

```
min_price = min(zip(prices.values(), prices.keys()))
# min_price is (10.75, 'FB')
```

```
max_price = max(zip(prices.values(), prices.keys()))
# max_price is (612.78, 'AAPL')
```

sorted() returns a list of tuples where each tuple is the key-value pair from the dictionary, sorted by the first element of the tuple (the price).

```
prices_sorted = sorted(zip(prices.values(), prices.keys()))
# prices_sorted is [(10.75, 'FB'), (37.2, 'HPQ'),
#                  (45.23, 'ACME'), (205.55, 'IBM'),
#                  (612.78, 'AAPL')]
```

zip() returns a zip object that is an iterator of tuples where each tuple is the key-value pair from the dictionary.

```

prices_and_names = zip(prices.values(), prices.keys())
print

(min(prices_and_names))    # OK

print

(max(prices_and_names))    # ValueError: max() arg is an empty
sequence

```

## 1.8.3 字典

字典是Python中一种可变的容器，用于存储键值对。字典的键必须是不可变的，而值可以是任何对象。字典的语法如下：

```
{key: value, key: value, ...}
```

```

min(prices)    # Returns 'AAPL'
max(prices)    # Returns 'IBM'

```

字典的键值对可以通过字典的 `values()` 方法获取。字典的语法如下：

```
prices.values()
```

```

min(prices.values())    # Returns 10.75

max(prices.values())    # Returns 612.78

```

如何找出字典中键值对的最小值或最大值？

使用 `key` 参数配合 `min()` 或 `max()` 函数即可。

```
min(prices, key=lambda  
k: prices[k])    # Returns 'FB'  
  
max(prices, key=lambda  
k: prices[k])    # Returns 'AAPL'
```

如何找出字典中的最小值或最大值？

```
min_value = prices[min(prices, key=lambda  
k: prices[k])]
```

使用 `zip()` 函数可以找出字典中的最小值或最大值。

使用 `zip()` 函数可以找出字典中的最小值或最大值。

value, key  
value key  
min() max() value  
key

```
>>> prices = { 'AAA' : 45.23, 'ZZZ': 45.23 }  
>>> min(zip(prices.values(), prices.keys()))  
(45.23, 'AAA')  
>>> max(zip(prices.values(), prices.keys()))  
(45.23, 'ZZZ')  
>>>
```

## 1.9

### 1.9.1

### 1.9.2

```
a = {  
    'x' : 1,  
    'y' : 2,  
    'z' : 3  
}  
  
b = {
```

```
'w' : 10,  
'x' : 11,  
'y' : 2  
}
```

keys() items()  
keys() items()

```
# Find keys in common  
a.keys() & b.keys() # { 'x', 'y' }  
  
# Find keys in a that are not in b  
a.keys() - b.keys() # { 'z' }  
  
# Find (key,value) pairs in common  
a.items() & b.items() # { ('y', 2) }
```

keys() items()  
keys() items()

```
# Make a new dictionary with certain keys removed  
c = {key:a[key] for  
key in  
a.keys() - {'z', 'w'}}  
# c is {'x': 1, 'y': 2}
```

## 1.9.3

```

keys()
keys-view
keys-view
keys-view
keys-view

```

```

    items().forEach((key,value) -> items-
view{key,value})
    }
}

```

values()

## 1.10

### 1.10.1 □□

[illegible]

## 1.10.2

```

    [] [] [] [] [] [] [] [] [] [] hashable [] [] [] [] [] [] [] [] [] []
    [] [] [] [] [] [] [] [] [] [] [] [] [] [] [1] []

```



```

item in
items:
    val = item if
key is
None else
key(item)
    if
val not in
seen:
    yield
item
    seen.add(val)

```

key

```

>>> a = [ {'x':1, 'y':2}, {'x':1, 'y':3}, {'x':1, 'y':2},
{'x':2, 'y':4}]
>>> list(dedupe(a, key=lambda
d: (d['x'],d['y'])))
[{'x': 1, 'y': 2}, {'x': 1, 'y': 3}, {'x': 2, 'y': 4}]
>>> list(dedupe(a, key=lambda
d: d['x']))
[{'x': 1, 'y': 2}, {'x': 2, 'y': 4}]
>>>

```



□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□

## 1.10.3 □□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□

```
>>> a  
[1, 5, 2, 1, 9, 1, 5, 10]  
>>> set(a)  
{1, 2, 10, 5, 9}  
>>>
```

□□□□□□□□□□□□□□□□□□□□ [2] □□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□——□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
with  
open(somefile,'r') as  
f:  
    for  
line in  
dedupe(f):
```

...

用dedupe()函数和sorted()、  
min()和max()的key参数来找出1.8和  
1.13

## 1.11 字典

### 1.11.1 字典

字典是Python中最常用的数据结构之一，  
它由键值对组成。

### 1.11.2 字典

字典的键必须是不可变的，而值可以是任意的。  
字典的键值对用方括号[]表示。

```
#####  
012345678901234567890123456789012345678901234567890'  
record = '.....100 .....513.25 .....'  
cost = int(record[20:32]) * float(record[40:48])
```

字典的键值对用方括号[]表示。

```
SHARES = slice(20,32)
PRICE = slice(40,48)

cost = int(record[SHARES]) * float(record[PRICE])
```

### 1.11.3 ☐ ☐

slice()

```
>>> items = [0, 1, 2, 3, 4, 5, 6]
>>> a = slice(2, 4)
>>> items[2:4]
[2, 3]
>>> items[a]
[2, 3]
>>> items[a] = [10,11]
>>> items
[0, 1, 10, 11, 4, 5, 6]
>>> del

items[a]
>>> items
[0, 1, 4, 5, 6]
```

slice() returns a slice object with attributes s.start, s.stop, and s.step that can be used to find out the values of the various parameters.

```
>>> a = slice(
>>> a.start
10
>>> a.stop
50
>>> a.step
2
>>>
```

indices(size) returns a tuple containing the values of the parameters (start, stop, step) that would be used to slice the array with the same shape and stride as the original array. IndexError is raised if the start or stop values are out of bounds.

```
>>> s = 'HelloWorld'
>>> a.indices(len(s))
(5, 10, 2)
>>> for
i in
range(*a.indices(len(s))):
... print
(s[i])
...

w
r
```

```
d
>>>
```

## 1.12 字典的遍历

### 1.12.1 遍历字典

遍历字典的键、值、键值对

### 1.12.2 字典的遍历

collections模块的Counter类可以统计字典中每个键出现的次数。most\_common()方法返回字典中键值对最多的前n个键值对。

遍历字典的键、值、键值对

```
words = [
    'look', 'into', 'my', 'eyes', 'look', 'into', 'my', 'eyes',
    'the', 'eyes', 'the', 'eyes', 'the', 'eyes', 'not',
    'around', 'the',
    'eyes', "don't", 'look', 'around', 'the', 'eyes', 'look',
    'into',
    'my', 'eyes', "you're", 'under'
]

from collections import
Counter
```

```
word_counts = Counter(words)
top_three = word_counts.most_common(3)
print
(top_three)
# Outputs [('eyes', 8), ('the', 5), ('look', 4)]
```

### 1.12.3 ☐ ☐

```

    Counter
    Counter

```

```
>>> word_counts['not']
1
>>> word_counts['eyes']
8
>>>
```

```
>>> morewords =
['why', 'are', 'you', 'not', 'looking', 'in', 'my', 'eyes']
>>> for
word in
morewords:
...
word_counts[word] += 1
...
```

```
>>> word_counts['eyes']
9
>>>
```

□□□□□□□□update()□□□

```
>>> word_counts.update(morewords)
>>>
```

Counter

```
>>> a = Counter(words)
>>> b = Counter(morewords)
>>> a
Counter({'eyes': 8, 'the': 5, 'look': 4, 'into': 3, 'my': 3,
'around': 2,
        "you're": 1, "don't": 1, 'under': 1, 'not': 1})
>>> b
Counter({'eyes': 1, 'looking': 1, 'are': 1, 'in': 1, 'not': 1,
'you': 1,
        'my': 1, 'why': 1})

>>> # Combine counts

>>> c = a + b
>>> c
Counter({'eyes': 9, 'the': 5, 'look': 4, 'my': 4, 'into': 3,
'not': 2,
        'around': 2, "you're": 1, "don't": 1, 'in': 1, 'why':
1,
        'looking': 1, 'are': 1, 'under': 1, 'you': 1})

>>> # Subtract counts
```

```
>>> d = a - b
>>> d
Counter({'eyes': 7, 'the': 5, 'look': 4, 'into': 3, 'my': 2,
        'around': 2,
        "you're": 1, "don't": 1, 'under': 1})
>>>
```

Counter

## 1.13

### 1.13.1

### 1.13.2

operator itemgetter

```
rows = [
    {'fname': 'Brian', 'lname': 'Jones', 'uid': 1003},
    {'fname': 'David', 'lname': 'Beazley', 'uid': 1002},
    {'fname': 'John', 'lname': 'Cleese', 'uid': 1001},
    {'fname': 'Big', 'lname': 'Jones', 'uid': 1004}
```



```
]

```

rows\_by\_fname = sorted(rows, key=itemgetter('fname'))  
rows\_by\_uid = sorted(rows, key=itemgetter('uid'))  
print(rows\_by\_fname)  
print(rows\_by\_uid)

```
from operator import itemgetter

rows_by_fname = sorted(rows, key=itemgetter('fname'))
rows_by_uid = sorted(rows, key=itemgetter('uid'))

print(rows_by_fname)
print(rows_by_uid)
```

rows\_by\_fname

```
[{'fname': 'Big', 'uid': 1004, 'lname': 'Jones'},
 {'fname': 'Brian', 'uid': 1003, 'lname': 'Jones'},
 {'fname': 'David', 'uid': 1002, 'lname': 'Beazley'},
 {'fname': 'John', 'uid': 1001, 'lname': 'Cleese'}]

[{'fname': 'John', 'uid': 1001, 'lname': 'Cleese'},
 {'fname': 'David', 'uid': 1002, 'lname': 'Beazley'},
 {'fname': 'Brian', 'uid': 1003, 'lname': 'Jones'},
 {'fname': 'Big', 'uid': 1004, 'lname': 'Jones'}]
```

itemgetter()

```
rows_by_lfname = sorted(rows, key=itemgetter('lname', 'fname'))
print
(rows_by_lfname)
```



```
rows_by_lfname = sorted(rows, key=lambda  
r: (r['lname'],r['fname']))
```

itemgetter() 是 Python 2.3 版本引入的一个函数，它返回一个 callable 对象，这个 callable 对象可以接受一个序列作为参数，并返回一个元组，元组的元素是序列中指定位置的元素。

min() 和 max() 函数可以接受一个 callable 对象作为 key 参数，用于指定排序的键。

```
>>> min(rows, key=itemgetter('uid'))  
{'fname': 'John', 'lname': 'Cleese', 'uid': 1001}  
>>> max(rows, key=itemgetter('uid'))  
{'fname': 'Big', 'lname': 'Jones', 'uid': 1004}  
>>>
```

## 1.14 字典的排序

### 1.14.1 按值排序

字典的排序可以通过 sorted() 函数实现，sorted() 函数可以接受一个 callable 对象作为 key 参数，用于指定排序的键。

### 1.14.2 按键排序

sorted() 函数可以接受一个 callable 对象作为 key 参数，用于指定排序的键。callable 对象是指可以调用（即具有 \_\_call\_\_ 方法）的对象。

sorted()를 사용하면 리스트의 요소를 정렬할 수 있다.  
User 클래스의 user\_id 속성을 기준으로 정렬하는 예제이다.  
sorted()의 key 인자로 User 클래스의 user\_id 속성을 지정한다.

```
>>> class User:
...     def __init__(self, user_id):
...         self.user_id = user_id
...     def __repr__(self):
...         return 'User({})'.format(self.user_id)
...
>>> users = [User(23), User(3), User(99)]
>>> users
[User(23), User(3), User(99)]
>>> sorted(users, key=lambda u: u.user_id)
[User(3), User(23), User(99)]
>>>
```

lambda 함수와 operator.attrgetter()를 사용하여 정렬할 수 있다.

```
>>> from operator import attrgetter
>>> sorted(users, key=attrgetter('user_id'))
[User(3), User(23), User(99)]
>>>
```

### 1.14.3 defaultdict

lambda 함수와 attrgetter()를 사용하여 defaultdict를 만들 수 있다.  
operator.itemgetter()를 사용하여 defaultdict의 값을 가져올 수 있다.

1.13 `User` objects  
`first_name` `last_name`

```
by_name = sorted(users, key=attrgetter('last_name',  
    'first_name'))
```

`min()` `max()`

```
>>> min(users, key=attrgetter('user_id'))  
User(3)  
>>> max(users, key=attrgetter('user_id'))  
User(99)  
>>>
```

## 1.15

### 1.15.1

### 1.15.2

`itertools.groupby()`

```
rows = [
    {'address': '5412 N CLARK', 'date': '07/01/2012'},
    {'address': '5148 N CLARK', 'date': '07/04/2012'},
    {'address': '5800 E 58TH', 'date': '07/02/2012'},
    {'address': '2122 N CLARK', 'date': '07/03/2012'},
    {'address': '5645 N RAVENSWOOD', 'date': '07/02/2012'},
    {'address': '1060 W ADDISON', 'date': '07/02/2012'},
    {'address': '4801 N BROADWAY', 'date': '07/01/2012'},
    {'address': '1039 W GRANVILLE', 'date': '07/04/2012'},
]
```

如何按日期对数据进行分组  
 使用 `date` 字段进行分组  
`itertools.groupby()`

```
from operator import itemgetter
from itertools import groupby

# Sort by the desired field first

rows.sort(key=itemgetter('date'))

# Iterate in groups

for date, items in groupby(rows, key=itemgetter('date')):
    print(date)
    for i in items:
        print(' ', i)
```

输出结果

```
07/01/2012
    {'date': '07/01/2012', 'address': '5412 N CLARK'}
    {'date': '07/01/2012', 'address': '4801 N BROADWAY'}
```

```

07/02/2012
    {'date': '07/02/2012', 'address': '5800 E 58TH'}
    {'date': '07/02/2012', 'address': '5645 N RAVENSWOOD'}
    {'date': '07/02/2012', 'address': '1060 W ADDISON'}
07/03/2012
    {'date': '07/03/2012', 'address': '2122 N CLARK'}
07/04/2012
    {'date': '07/04/2012', 'address': '5148 N CLARK'}
    {'date': '07/04/2012', 'address': '1039 W GRANVILLE'}

```

## 1.15.3 defaultdict

`groupby()` returns an iterator of `key` and `groupby()` returns an iterator of `value` and `sub_iterator`

`groupby()` returns an iterator of `key` and `groupby()` returns an iterator of `value`

`defaultdict()` returns an iterator of `key` and `multidict`

```

from collections import defaultdict
rows_by_date = defaultdict(list)
for row in rows:
    rows_by_date[row['date']].append(row)

```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> for
r in
rows_by_date['07/01/2012']:
...     print
(r)
...

{'date': '07/01/2012', 'address': '5412 N CLARK'}
{'date': '07/01/2012', 'address': '4801 N BROADWAY'}
>>>
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□groupby()□□□□□□□□□□

## 1.16 □□□□□□□□□□

### 1.16.1 □□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□

### 1.16.2 □□□□



# list comprehension

```
>>> mylist = [1, 4, -5, 10, -7, 2, 3, -1]
>>> [n for
n in
mylist if
n > 0]
[1, 4, 10, 2, 3]
>>> [n for
n in
mylist if
n < 0]
[-5, -7, -1]
>>>
```

```


```

```
>>> pos = (n for
n in
mylist if
n > 0)
>>> pos
at 0x1006a0eb0>
>>> for
```

```

x in
pos:
...     print
(x)
...

1
4
10
2
3
>>>

```

```

filter()

```

```

values = ['1', '2', '-3', '-', '4', 'N/A', '5']

def
is_int(val):
    try
:
        x = int(val)
        return
True
    except ValueError
:
        return

```

```
False
```

```
ivals = list(filter(is_int, values))  
print
```

```
(ivals)  
# Outputs ['1', '2', '-3', '4', '5']
```

`filter()` returns an iterator that filters elements from an iterable. The `list()` function is used to convert the iterator to a list.

## 1.16.3 `filter()`

The `filter()` function filters elements from an iterable based on a function. The function is applied to each element, and if it returns a truthy value, the element is included in the resulting iterator.

```
>>> mylist = [1, 4, -5, 10, -7, 2, 3, -1]  
>>> import math  
  
>>> [math.sqrt(n) for  
n in  
mylist if  
n > 0]  
[1.0, 2.0, 3.1622776601683795, 1.4142135623730951,  
1.7320508075688772]  
>>>
```

itertools.compress() 函数可以接受一个可迭代对象，以及一个布尔值。如果布尔值为 True，则返回原可迭代对象；如果布尔值为 False，则返回一个空列表。

```
>>> clip_neg = [n if
n > 0 else
0 for
n in
mylist]
>>> clip_neg
[1, 4, 0, 10, 0, 2, 3, 0]
>>> clip_pos = [n if
n < 0 else
0 for
n in
mylist]
>>> clip_pos
[0, 0, -5, 0, -7, 0, 0, -1]
>>>
```

itertools.compress() 函数可以接受一个可迭代对象，以及一个布尔值。如果布尔值为 True，则返回原可迭代对象；如果布尔值为 False，则返回一个空列表。

```
addresses = [
    '5412 N CLARK',
    '5148 N CLARK',
    '5800 E 58TH',
    '2122 N CLARK',
    '5645 N RAVENSWOOD',
    '1060 W ADDISON',
    '4801 N BROADWAY',
    '1039 W GRANVILLE',
]

counts = [ 0, 3, 10, 4, 1, 7, 6, 1]
```

0 3 10 4 1 7 6 1 count 5  
 0 3 10 4 1 7 6 1

```
>>> from itertools import compress
>>> more5 = [n > 5 for n in counts]
>>> more5
[False, False, True, False, False, True, True, False]
>>> list(compress(addresses, more5))
['5800 E 58TH', '4801 N BROADWAY', '1039 W GRANVILLE']
>>>
```

0 3 10 4 1 7 6 1 count 5  
 0 3 10 4 1 7 6 1 compress() True

filter() compress()  
 list()

## 1.17

## 1.17.1

## 1.17.2

dictionary comprehension

```
prices = {
    'ACME': 45.23,
    'AAPL': 612.78,
    'IBM': 205.55,
    'HPQ': 37.20,
    'FB': 10.75
}

# Make a dictionary of all prices over 200

p1 = { key:value for
key, value in
prices.items() if
value > 200 }

# Make a dictionary of tech stocks

tech_names = { 'AAPL', 'IBM', 'HPQ', 'MSFT' }
p2 = { key:value for
key,value in
prices.items() if
```

```
key in
tech_names }
```

## 1.17.3 dict()

dict() is a function that creates a dictionary. It can take a list of key-value pairs as input, or a single iterable of key-value pairs. In this case, we are using it to create a dictionary from a list of key-value pairs.

```
p1 = dict((key, value) for
key, value in
prices.items() if
value > 200)
```

dict() can also take a single iterable of key-value pairs as input. In this case, we are using it to create a dictionary from a list of key-value pairs.

dict() can also take a single iterable of key-value pairs as input. In this case, we are using it to create a dictionary from a list of key-value pairs.

```
# Make a dictionary of tech stocks
tech_names = { 'AAPL', 'IBM', 'HPQ', 'MSFT' }
p2 = { key:prices[key] for
key in
prices.keys() & tech_names }
```

Python 1.6  
Python 14.13

## 1.18 Python

### 1.18.1

Python

### 1.18.2

collections.namedtuple()  
collections.namedtuple()  
Python

```
>>> from collections import namedtuple
>>> Subscriber = namedtuple('Subscriber', ['addr', 'joined'])
>>> sub = Subscriber('jonesy@example.com', '2012-10-19')
>>> sub
Subscriber(addr='jonesy@example.com', joined='2012-10-19')
```



```
>>> sub.addr
'jonesy@example.com'
>>> sub.joined
'2012-10-19'
>>>
```

namedtuple  
indexing  
unpacking

```
>>> len(sub)
2
>>> addr, joined = sub
>>> addr
'jonesy@example.com'
>>> joined
'2012-10-19'
>>>
```

```
def
compute_cost(records):
    total = 0.0
    for
```

```

rec in
records:
    total += rec[1] * rec[2]
    return
total

```

1.18.2 使用 namedtuple 计算股票成本

```

from collections import namedtuple

Stock = namedtuple('Stock', ['name', 'shares', 'price'])
def compute_cost(records):
    total = 0.0
    for rec in records:
        s = Stock(*rec)
        total += s.shares * s.price
    return total

```

1.18.3 使用 namedtuple 计算股票成本

### 1.18.3 使用 namedtuple 计算股票成本

namedtuple 是一个类，它允许你创建具有名称的元组。

namedtuple 的创建方法如下：

namedtuple('Stock', ['name', 'shares', 'price'])

namedtuple 返回一个 immutable 的类。

```
>>> s = Stock('ACME', 100, 123.45)
>>> s
Stock(name='ACME', shares=100, price=123.45)
>>> s.shares = 75
Traceback (most recent call last):
  File "<stdin>", line 1, in
<module>
AttributeError

: can't set attribute
>>>
```

namedtuple 的 `_replace()` 方法可以创建一个新的 namedtuple 实例，其中指定的属性值被替换。

```
>>> s = s._replace(shares=75)
>>> s
Stock(name='ACME', shares=75, price=123.45)
>>>
```

`_replace()` 方法返回一个新的 namedtuple 实例，其中指定的属性值被替换。这允许在不修改原始实例的情况下创建新的实例。

“” 方法 `_replace()` 可以创建新的 namedtuple 实例。

```
from collections import namedtuple

Stock = namedtuple('Stock', ['name', 'shares', 'price',
                              'date', 'time'])

# Create a prototype instance
```

```
stock_prototype = Stock('', 0, 0.0, None, None)

# Function to convert a dictionary to a Stock

def dict_to_stock(s):
    return stock_prototype._replace(**s)
```

```
>>> a = {'name': 'ACME', 'shares': 100, 'price': 123.45}
>>> dict_to_stock(a)
Stock(name='ACME', shares=100, price=123.45, date=None,
time=None)
>>> b = {'name': 'ACME', 'shares': 100, 'price': 123.45,
'date': '12/17/2012'}
>>> dict_to_stock(b)
Stock(name='ACME', shares=100, price=123.45,
date='12/17/2012', time=None)
>>>
```

```

    namedtuple(
        __slots__, 8.4
    )

```

**1.19** □□□□□□□□□□

### 1.19.1 □□

reductionsum()  
min()max()

## 1.19.2

——

```
nums = [1, 2, 3, 4, 5]
s = sum(x * x for
x in
nums)
```

```
# Determine if any .py files exist in a directory
import os

files = os.listdir('dirname')
if
any(name.endswith('.py') for
name in
files):
    print
('There be python!')
else
:
```

```

    print
('Sorry, no python.')

# Output a tuple as CSV
s = ('ACME', 50, 123.45)
print
(',', '.join(str(x) for
x in
s))

# Data reduction across fields of a data structure
portfolio = [
    {'name': 'GOOG', 'shares': 50},
    {'name': 'YHOO', 'shares': 75},
    {'name': 'AOL', 'shares': 20},
    {'name': 'SCOX', 'shares': 65}
]
min_shares = min(s['shares'] for
s in
portfolio)

```

## 1.19.3

1.19.3

```

s = sum((x * x for
x in

```

```
nums))    # Pass generator-expr as argument
s = sum(x * x for
x in
nums)     # More elegant syntax
```

```


```

```
nums = [1, 2, 3, 4, 5]
s = sum([x * x for
x in
nums])
```

```


```

```


```

```
# Original: Returns 20
min_shares = min(s['shares'] for
s in
```

```
portfolio)

# Alternative: Returns {'name': 'AOL', 'shares': 20}
min_shares = min(portfolio, key=lambda
s: s['shares'])
```

## 1.20 字典的遍历

### 1.20.1 字典的键

字典的键是字典中唯一的标识符，用于访问字典中的值。键可以是字符串、数字、元组等不可变对象。键的值可以是任意对象。

### 1.20.2 字典的值

字典的值是字典中键所指向的对象。值可以是任意对象。

```
a = {'x': 1, 'z': 3 }
b = {'y': 2, 'z': 4 }
```

字典的遍历是指遍历字典中的键、值或键值对。Python 提供了多种方法来遍历字典。例如，使用 `keys()` 方法可以遍历字典的键，使用 `values()` 方法可以遍历字典的值，使用 `items()` 方法可以遍历字典的键值对。此外，还可以使用 `collections.ChainMap` 来合并多个字典并遍历。

```
from collections import ChainMap
c = ChainMap(a,b)
print(c['x']) # Outputs 1 (from a)
```



```
print(c['y']) # Outputs 2 (from b)
```

```
print(c['z']) # Outputs 3 (from a)
```

## 1.20.3 ChainMap

ChainMap은 여러 Mapping 객체를 단일 Mapping 객체로 결합하는 데 사용됩니다. ChainMap은 Mapping 객체의 순서를 유지하며, 첫 번째 Mapping 객체에서 값을 찾지 못하면 다음 Mapping 객체를 시도합니다. ChainMap은 Mapping 객체의 순서를 유지하며, 첫 번째 Mapping 객체에서 값을 찾지 못하면 다음 Mapping 객체를 시도합니다.

```
>>> len(c)
3
>>> list(c.keys())
['x', 'y', 'z']
>>> list(c.values())
[1, 2, 3]
>>>
```

ChainMap은 Mapping 객체의 순서를 유지하며, 첫 번째 Mapping 객체에서 값을 찾지 못하면 다음 Mapping 객체를 시도합니다. ChainMap은 Mapping 객체의 순서를 유지하며, 첫 번째 Mapping 객체에서 값을 찾지 못하면 다음 Mapping 객체를 시도합니다.

ChainMap은 Mapping 객체의 순서를 유지하며, 첫 번째 Mapping 객체에서 값을 찾지 못하면 다음 Mapping 객체를 시도합니다. ChainMap은 Mapping 객체의 순서를 유지하며, 첫 번째 Mapping 객체에서 값을 찾지 못하면 다음 Mapping 객체를 시도합니다.

```
>>> c['z'] = 10
>>> c['w'] = 40
```

```

>>> del

c['x']
>>> a
{'w': 40, 'z': 10}
>>> del

c['y']
Traceback (most recent call last):
...

KeyError: "Key not found in the first mapping: 'y'"
>>>

```

## ChainMap

ChainMap is a class that provides a way to create a new mapping that is a combination of two or more mappings. It is useful for creating a new mapping that is a combination of two or more mappings, or for creating a new mapping that is a combination of two or more mappings.

```

>>> values = ChainMap()
>>> values['x'] = 1
>>> # Add a new mapping

>>> values = values.new_child()
>>> values['x'] = 2
>>> # Add a new mapping

>>> values = values.new_child()
>>> values['x'] = 3
>>> values
ChainMap({'x': 3}, {'x': 2}, {'x': 1})
>>> values['x']
3
>>> # Discard last mapping

>>> values = values.parents

```

```

>>> values['x']
2
>>> # Discard last mapping

>>> values = values.parents
>>> values['x']
1
>>> values
ChainMap({'x': 1})
>>>

```

ChainMap() update()

```

>>> a = {'x': 1, 'z': 3 }
>>> b = {'y': 2, 'z': 4 }
>>> merged = dict(b)
>>> merged.update(a)
>>> merged['x']
1
>>> merged['y']
2
>>> merged['z']
3
>>>

```

ChainMap() update()

```

>>> a['x'] = 13
>>> merged['x']
1

```

## ChainMap

```
>>> a = {'x': 1, 'z': 3 }
>>> b = {'y': 2, 'z': 4 }
>>> merged = ChainMap(a, b)
>>> merged['x']
1
>>> a['x'] = 42
>>> merged['x']    # Notice change to merged dicts

42
>>>
```

[1] `__hash__()`

[2]

[3] `flat file`

[4] `records`  
`shares`  
`price`



## 第2章 字符串

字符串是Python中最常用的数据类型之一。它是由一系列字符组成的序列，可以包含字母、数字、符号等。字符串是不可变的，一旦创建就不能修改。Python中的字符串支持多种操作，如索引、切片、拼接等。Unicode是国际标准的字符编码，Python 3默认使用Unicode编码。

## 2.1 字符串的基本操作

### 2.1.1 字符串的创建

字符串可以通过单引号或双引号来创建。单引号和双引号在Python中是等价的，都可以用来创建字符串。例如，可以创建以下字符串：

### 2.1.2 字符串的索引

字符串支持索引操作，可以通过索引来访问字符串中的单个字符。索引从0开始，到字符串的最后一个字符结束。Python还支持切片操作，可以通过切片来访问字符串的一部分。此外，Python还提供了正则表达式模块，可以通过`re.split()`来按正则表达式分割字符串。

```
>>> line = 'asdf fjdk; afed, fjek,asdf,      foo'
>>> import re
```

```
>>> re.split(r'[;,\s]\s*', line)
['asdf', 'fjdk', 'afed', 'fjek', 'asdf', 'foo']
```

## 2.1.3 分割

`re.split()` 函数返回一个列表，列表中的每个元素都是一个字符串，这些字符串是由正则表达式分割的。如果正则表达式包含捕获组，那么捕获组的内容也会包含在列表中。与 `str.split()` 函数相比，`re.split()` 函数可以处理更复杂的分割规则。

使用 `re.split()` 函数时，正则表达式中的捕获组会被包含在结果列表中。例如，使用正则表达式 `(;|,|\s)\s*` 分割字符串 `'asdf; fjdk; afed; fjek; asdf; foo'`，结果列表中的每个元素都是一个字符串，这些字符串是由正则表达式分割的。如果正则表达式包含捕获组，那么捕获组的内容也会包含在列表中。

```
>>> fields = re.split(r'(;|,|\s)\s*', line)
>>> fields
['asdf', ' ', 'fjdk', ';', 'afed', ', ', 'fjek', ', ', 'asdf',
', ', 'foo']
>>>
```

在上面的例子中，正则表达式 `(;|,|\s)\s*` 包含捕获组，因此捕获组的内容也会被包含在结果列表中。为了去除捕获组的内容，可以使用非捕获组 `(?:;|,|\s)\s*`。使用非捕获组分割字符串 `'asdf; fjdk; afed; fjek; asdf; foo'`，结果列表中的每个元素都是一个字符串，这些字符串是由正则表达式分割的。非捕获组不会包含在结果列表中。

```
>>> values = fields[::2]
>>> delimiters = fields[1::2] + ['']
>>> values
['asdf', 'fjdk', 'afed', 'fjek', 'asdf', 'foo']
>>> delimiters
[' ', ';', ', ', ', ', ', ', ', '']
```

```
>>> # Reform the line using the same delimiters

>>> ''.join(v+d for
v,d in
zip(values, delimiters))
'asdf fjdk;afed,fjek,asdf,foo'
>>>
```

1. 使用 `zip` 函数将 `values` 和 `delimiters` 列表打包成元组列表。
 2. 使用 `join` 方法将元组列表中的每个元组中的元素连接起来，使用空字符串 `''` 作为分隔符。

```
>>> re.split(r'(?:,|;\s)\s*', line)
['asdf', 'fjdk', 'afed', 'fjek', 'asdf', 'foo']
>>>
```

## 2.2 正则表达式

### 2.2.1 正则表达式

1. 正则表达式是一种强大的文本处理工具，可以用于匹配、查找、替换等操作。
 2. 正则表达式由一系列字符组成，这些字符按照一定的规则组合起来，用于描述文本的模式。

### 2.2.2 正则表达式



字符串的startswith()和endswith()方法

```
>>> filename = 'spam.txt'
>>> filename.endswith('.txt')
True
>>> filename.startswith('file:')
False
>>> url = 'http://www.python.org'
>>> url.startswith('http:')
True
>>>
```

字符串的startswith()和endswith()方法

```
>>> import os
>>> filenames = os.listdir('.')
>>> filenames
[ 'Makefile', 'foo.c', 'bar.py', 'spam.c', 'spam.h' ]
>>> [name for name in filenames if name.endswith(('c', 'h'))]
[ 'foo.c', 'spam.c', 'spam.h' ]
>>> any(name.endswith('.py') for name in filenames)
True
>>>
```

字符串的startswith()和endswith()方法

```
from urllib.request import urlopen

def read_data(name):
    if name.startswith(('http:', 'https:', 'ftp:')):
        return urlopen(name).read()
```

```
else:
    with open(name) as f:
        return f.read()
```

Python에서 tuple() 함수를 사용하여 리스트를 튜플로 변환할 수 있습니다.

```
>>> choices = ['http:', 'ftp:']
>>> url = 'http://www.python.org'
>>> url.startswith(choices)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: startswith first arg must be str or a tuple of str, not list
>>>

url.startswith(tuple(choices))
True
>>>
```

## 2.2.3 문자열 슬라이싱

startswith()와 endswith() 함수는 문자열이 특정 문자열로 시작하거나 끝나는지 여부를 확인합니다.

```
>>> filename = 'spam.txt'
>>> filename[-4:] == '.txt'
True
>>> url = 'http://www.python.org'
>>> url[:5] == 'http:' or
url[:6] == 'https:' or
```

```
url[:4] == 'ftp:'  
True  
>>>
```

正则表达式

```
>>> import re  
>>> url = 'http://www.python.org'  
>>> re.match('http:|https:|ftp:', url)  
<_sre.SRE_Match object at 0x101253098>  
>>>
```

正则表达式

正则表达式

```
if  
any(name.endswith(('c', 'h')) for  
name in  
listdir(dirname)):  
...
```

## 2.3 Shell

## 2.3.1 通配符

在UNIX Shell中，通配符用于匹配文件名。例如，`*.py`匹配所有Python文件，`Dat[0-9]*.csv`匹配所有以`Dat`开头，后跟一个数字，再以`.csv`结尾的文件。

## 2.3.2 文件匹配

`fnmatch`模块提供了与UNIX Shell中通配符匹配相同的函数：`fnmatch()`和`fnmatchcase()`。前者区分大小写，后者不区分大小写。

```
>>> from fnmatch import fnmatch, fnmatchcase
>>> fnmatch('foo.txt', '*.txt')
True
>>> fnmatch('foo.txt', '?oo.txt')
True
>>> fnmatch('Dat45.csv', 'Dat[0-9]*')
True
>>> names = ['Dat1.csv', 'Dat2.csv', 'config.ini', 'foo.py']
>>> [name for
name in
names if
fnmatch(name, 'Dat*.csv')]
['Dat1.csv', 'Dat2.csv']
>>>
```

`fnmatch()`和`fnmatchcase()`的语法如下：

```
>>> # On OS X (Mac)
>>> fnmatch('foo.txt', '*.TXT')
```

False

```
>>> # On Windows
>>> fnmatch('foo.txt', '*.TXT')
True
>>>
```

fnmatchcase()

```
>>> fnmatchcase('foo.txt', '*.TXT')
False
>>>
```

addresses = [

```
    '5412 N CLARK ST',
    '1060 W ADDISON ST',
    '1039 W GRANVILLE AVE',
    '2122 N CLARK ST',
    '4802 N BROADWAY',
]
```

from fnmatch import fnmatchcase

```
>>> from fnmatch import fnmatchcase
>>> [addr for addr in addresses if fnmatchcase(addr, '* ST')]
['5412 N CLARK ST', '1060 W ADDISON ST', '2122 N CLARK ST']
>>> [addr for addr in addresses if fnmatchcase(addr, '54[0-9]
[0-9] *CLARK*')]
['5412 N CLARK ST']
```

```
>>>
```

## 2.3.3 fnmatch

fnmatch 模块提供了与 Unix shell 文件命名模式匹配的函数。它包含以下函数：

fnmatch.fnmatch(pattern, string) 检查字符串是否与模式匹配。glob 模块提供了与 Unix shell 文件命名模式匹配的函数。它包含以下函数：

## 2.4 字符串操作

### 2.4.1 字符串

字符串是不可变的序列。它们由字符组成。

### 2.4.2 字符串方法

字符串对象有以下方法：

- str.find() 返回子字符串在字符串中首次出现的位置。
- str.endswith() 检查字符串是否以指定的子字符串结尾。
- str.startswith() 检查字符串是否以指定的子字符串开头。

```
>>> text = 'yeah, but no, but yeah, but no, but yeah'
>>> # Exact match
```

```

>>> text == 'yeah'
False

>>> # Match at start or end

>>> text.startswith('yeah')
True
>>> text.endswith('no')
False

>>> # Search for the location of the first occurrence

>>> text.find('no')
10
>>>

```

re
   
 "11/27/2012"

```

>>> text1 = '11/27/2012'
>>> text2 = 'Nov 27, 2012'
>>>
>>> import re

>>> # Simple matching: \d+ means match one or more digits

>>> if
re.match(r'\d+/\d+/\d+', text1):
...     print
('yes')
... else

```





```

...

yes
>>> if

datepat.match(text2):
...     print

('yes')
... else

:
...     print

('no')
...

no
>>>

```

match() returns a match object if the pattern matches the text, otherwise it returns None. findall() returns a list of all matches.

```

>>> text = 'Today is 11/27/2012. PyCon starts 3/13/2013.'
>>> datepat.findall(text)
['11/27/2012', '3/13/2013']
>>>

```

re.compile() is used to compile a regular expression pattern into a regular expression object. This object is then used to match or find all occurrences of the pattern in a string.

```

>>> datepat = re.compile(r'(\d+)/(\d+)/(\d+)')
>>>

```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□

```
>>> m = datepat.match('11/27/2012')
>>> m

<_sre.SRE_Match object at 0x1005d2750>
>>> # Extract the contents of each group

>>> m.group(0)
'11/27/2012'
>>> m.group(1)
'11'
>>> m.group(2)
'27'
>>> m.group(3)
'2012'
>>> m.groups()
('11', '27', '2012')
>>> month, day, year = m.groups()
>>>

>>> # Find all matches (notice splitting into tuples)

>>> text
'Today is 11/27/2012. PyCon starts 3/13/2013.'
>>> datepat.findall(text)
[('11', '27', '2012'), ('3', '13', '2013')]
>>> for

month, day, year in

datepat.findall(text):
...     print

('{}-{}-{}'.format(year, month, day))
...

2012-11-27
2013-3-13
```

```
>>>
```

```
findall() returns a list of all the matches found in the text.
finditer() returns an iterator that yields match objects.
```

```
>>> for
m in
datepat.finditer(text):
...     print
(m.groups())
...

('11', '27', '2012')
('3', '13', '2013')
>>>
```

## 2.4.3 正则表达式

正则表达式 (Regular Expression) 是一种强大的文本处理工具。它允许你定义一个模式，用于匹配文本中的特定结构。在 Python 中，正则表达式通过 `re` 模块来实现。你可以使用 `re.compile()` 来编译一个正则表达式，然后使用 `match()` 来匹配文本。此外，`findall()` 和 `finditer()` 也是常用的函数。

例如，如果你想匹配一个日期格式，比如 `MM/DD/YYYY`，你可以使用以下正则表达式：

```
r'(\d+)/(\d+)/(\d+)'
```

正则表达式匹配日期格式  
r'\d{4}/\d{2}/\d{2}'

使用match()方法匹配日期格式  
返回匹配对象

```
>>> m = datepat.match('11/27/2012abcdef')
>>> m
<_sre.SRE_Match object at 0x1005d27e8>
>>> m.group()
'11/27/2012'
>>>
```

使用match()方法匹配日期格式  
返回匹配对象

```
>>> datepat = re.compile(r'\d{4}/\d{2}/\d{2}$')
>>> datepat.match('11/27/2012abcdef')
>>> datepat.match('11/27/2012')
<_sre.SRE_Match object at 0x1005d2750>
>>>
```

使用findall()方法匹配日期格式  
返回匹配对象列表

```
>>> re.findall(r'\d{4}/\d{2}/\d{2}', text)
[('11', '27', '2012'), ('3', '13', '2013')]
>>>
```

Python 2.5 版本中，字符串是不可变的。这意味着一旦你创建了一个字符串，就不能再改变它。如果你需要修改字符串，你必须创建一个新的字符串。这可能会导致内存使用量的增加，特别是在处理大量字符串时。因此，在设计程序时，应该考虑到这一点，避免不必要的字符串操作。

## 2.5 字符串操作

### 2.5.1 字符串

字符串是不可变的，这意味着一旦你创建了一个字符串，就不能再改变它。

### 2.5.2 字符串操作

字符串操作包括字符串的替换、查找、分割、连接等。其中，`str.replace()` 方法用于替换字符串中的子串。

```
>>> text = 'yeah, but no, but yeah, but no, but yeah'
>>> text.replace('yeah', 'yep')
'yep, but no, but yep, but no, but yep'
>>>
```

字符串的查找和替换操作通常使用正则表达式。正则表达式是一种强大的文本处理工具，可以用于匹配、查找、替换等操作。在 Python 中，正则表达式模块 `re` 提供了丰富的接口来支持这些操作。例如，使用 `re.sub()` 方法可以替换字符串中的匹配项。

```
>>> text = 'Today is 11/27/2012. PyCon starts 3/13/2013.'
>>> import re
```

```
>>> re.sub(r'(\d+)/(\d+)/(\d+)', r'\3-\1-\2', text)
'Today is 2012-11-27. PyCon starts 2013-3-13.'
>>>
```

sub() 1 2 3

```
>>> import re
>>> datepat = re.compile(r'(\d+)/(\d+)/(\d+)')
>>> datepat.sub(r'\3-\1-\2', text)
'Today is 2012-11-27. PyCon starts 2013-3-13.'
>>>
```

```
>>> from calendar import month_abbr
>>> def change_date(m):
...     mon_name = month_abbr[int(m.group(1))]
...     return '{} {} {}'.format(m.group(2), mon_name,
m.group(3))
...
>>> datepat.sub(change_date, text)
'Today is 27 Nov 2012. PyCon starts 13 Mar 2013.'
>>>
```

match() find() group()

re.subn() 返回一个元组，包含替换后的字符串和替换的次数。

```
>>> newtext, n = datepat.subn(r'\3-\1-\2', text)
>>> newtext
'Today is 2012-11-27. PyCon starts 2013-3-13.'
>>> n
2
>>>
```

## 2.5.3 re.sub()

re.sub() 函数与 re.subn() 函数类似，但返回的是替换后的字符串，而不是替换的次数。

## 2.6 re.compile()

### 2.6.1 re.compile()

re.compile() 函数用于编译正则表达式，返回一个 re 对象。

### 2.6.2 re.compile() 的 flags 参数

re.compile() 函数的 flags 参数用于指定正则表达式的编译选项。re.IGNORECASE 标志用于忽略大小写。

```

>>> text = 'UPPER PYTHON, lower python, Mixed Python'
>>> re.findall('python', text, flags=re.IGNORECASE)
['PYTHON', 'python', 'Python']
>>> re.sub('python', 'snake', text, flags=re.IGNORECASE)
'UPPER snake, lower snake, Mixed snake'
>>>

```

support  
 function

```

def
matchcase(word):
    def
    replace(m):
        text = m.group()
        if
    text.isupper():
        return
    word.upper()
    elif
    text.islower():
        return
    word.lower()
    elif
    text[0].isupper():
        return
    word.capitalize()
    else
    :

```



```
word
return
replace
```

```
>>> re.sub('python', matchcase('snake'), text,
flags=re.IGNORECASE)
'UPPER SNAKE, lower snake, Mixed Snake'
>>>
```

```
re.IGNORECASE
case
folding
Unicode
2.10
```

## 2.7.2 正则表达式

正则表达式（Regular Expression）是一种强大的文本处理工具，广泛应用于字符串匹配、搜索和替换等操作。它提供了一种简洁而灵活的方式来描述文本的模式。

```
>>> str_pat = re.compile(r'\"(.*)\"')
>>> text1 = 'Computer says "no."'
>>> str_pat.findall(text1)
['no.']
>>> text2 = 'Computer says "no." Phone says "yes."'
>>> str_pat.findall(text2)
['no." Phone says "yes.']
>>>
```

在上面的代码中，我们使用 `re.compile(r'\"(.*)\"')` 编译了一个正则表达式，用于匹配双引号内的任意字符串。然后，我们使用 `findall` 方法在 `text1` 和 `text2` 中查找匹配项。在 `text1` 中，只匹配到了 `"no."`；而在 `text2` 中，匹配到了 `"no." Phone says "yes."`，这显然不是我们想要的结果。

为了解决这个问题，我们需要使用非贪婪匹配。在正则表达式中，通过在量词后面添加一个问号（`?`），可以实现非贪婪匹配。例如，`\"(.*)?\"` 表示匹配最短的字符串。

```
>>> str_pat = re.compile(r'\"(.*)?\"')
>>> str_pat.findall(text2)
['no.', 'yes.']
>>>
```

通过修改正则表达式为 `\"(.*)?\"`，我们成功地实现了非贪婪匹配。现在，在 `text2` 中，`findall` 方法返回了 `['no.', 'yes.']`，这正是我们期望的结果。

## 2.7.3 正则表达式

正则表达式（Regular Expression）是一种强大的文本处理工具，广泛应用于字符串匹配、搜索、替换等操作。它通过一系列预定义的规则来描述文本的模式。正则表达式通常由普通字符（如字母、数字、下划线等）和特殊字符（如点、星号、括号等）组成。例如，正则表达式 `ab*c` 表示匹配以 `a` 开头，以 `c` 结尾，中间包含任意数量（0 个或多个）的 `b` 的字符串。

## 2.8 正则表达式的应用

### 2.8.1 字符串匹配

正则表达式可以用于匹配字符串中的特定模式。例如，使用正则表达式 `re.match(pattern, string)` 可以检查字符串 `string` 是否以模式 `pattern` 开头。如果匹配成功，返回一个匹配对象；否则返回 `None`。

### 2.8.2 字符串替换

正则表达式还可以用于替换字符串中的内容。使用 `re.sub(pattern, replacement, string)` 函数，可以将字符串 `string` 中所有匹配模式 `pattern` 的子串替换为 `replacement`。

```
>>> comment = re.compile(r'/*(.*?)*/')
>>> text1 = '/* this is a comment */'
>>> text2 = '''/* this is a
...
multiline comment */
...
'''
```

```
>>>
>>> comment.findall(text1)
[' this is a comment ']
>>> comment.findall(text2)
[]
>>>
```

正则表达式匹配多行注释

```
>>> comment = re.compile(r'/\*((?:.|\n)*?)\*/')
>>> comment.findall(text2)
[' this is a\n      multiline comment ']
>>>
```

正则表达式匹配多行注释

## 2.8.3 正则表达式

re.compile() 编译正则表达式——  
re.DOTALL 标志使正则表达式中的点 (.) 匹配任意字符，包括换行符。

```
>>> comment = re.compile(r'/\*((.*)?)\*/', re.DOTALL)
>>> comment.findall(text2)
[' this is a\n      multiline comment ']
```

正则表达式 re.DOTALL 标志使正则表达式中的点 (.) 匹配任意字符，包括换行符。

tokenizing  
tokenizing  
tokenizing

## 2.9 Unicode

### 2.9.1

Unicode  
Unicode

### 2.9.2

Unicode  
Unicode

```
>>> s1 = 'Spicy Jalape\u00f1'
0'
>>> s2 = 'Spicy Jalapen\u0303'
0'
>>> s1
'Spicy Jalape\u00f1o'
>>> s2
'Spicy Jalape\u00f1o'
>>> s1 == s2
False
>>> len(s1)
14
>>> len(s2)
15
```

```
>>>
```

看看“Spicy Jalapeño”是否完全由  
“ñ”完全组成  
U+00F1“n”“~”  
U+0303

看看是否完全由  
unicodedata

```
>>> import unicodedata
>>> t1 = unicodedata.normalize('NFC', s1)
>>> t2 = unicodedata.normalize('NFC', s2)
>>> t1 == t2
True
>>> print(ascii(t1))
'Spicy Jalape\xfl\u0303o'

>>> t3 = unicodedata.normalize('NFD', s1)
>>> t4 = unicodedata.normalize('NFD', s2)
>>> t3 == t4
True
>>> print(ascii(t3))
'Spicy Jalapen\u0303o'
>>>
```

normalize()  
NFC  
NFD

## Python NFKC NFKD

```
>>> s = '\ufb01'

' # A single character
>>> s
'fi'
>>> unicodedata.normalize('NFD', s)
'fi'

# Notice how the combined letters are broken apart here
>>> unicodedata.normalize('NFKD', s)
'fi'
>>> unicodedata.normalize('NFKC', s)
'fi'
>>>
```

### 2.9.3

Unicode

```
>>> t1 = unicodedata.normalize('NFD', s1)
>>> ''.join(c for
c in
t1 if not
```

```
unicodedata.combining(c))
'Spicy Jalapeno'
>>>
```

unicodedata.combining() 返回一个整数，表示该字符的 combining 属性。如果该字符没有 combining 属性，则返回 0。如果该字符有 combining 属性，则返回一个非零整数。该属性用于确定字符是否应该与前面的字符组合成一个字符。

Unicode 的归一化形式（Normalization）是指将不同的 Unicode 字符串转换为相同的字符串的过程。这通常用于比较字符串或存储字符串。

<http://www.unicode.org/faq/normalization.html> Ned Batchelder

<http://nedbatchelder.com/text/unipain.html> Python 的 Unicode 支持

## 2.10 Unicode

### 2.10.1

Unicode 是一个字符集，它定义了每个字符的编码。Unicode 的归一化形式是指将不同的 Unicode 字符串转换为相同的字符串的过程。

### 2.10.2



reUnicode  
\dUnicode

```
>>> import re
>>> num = re.compile('\d+')
>>> # ASCII digits

>>> num.match('123')
<_sre.SRE_Match object at 0x1007d9ed0>

>>> # Arabic digits

>>> num.match('\u0661\u0662\u0663')
<_sre.SRE_Match object at 0x101234030>
>>>
```

Unicode  
Unicode\uFFFF\UFFFFFF

```
>>> arabic = re.compile('[\u0600-\u06ff\u0750-\u077f\u08a0-\u08ff]+')
>>>
```

2.9  
case folding

```

>>> pat = re.compile('stra\u00df
e', re.IGNORECASE)
>>> s = 'stra'
>>> pat.match(s)                                # Matches

<_sre.SRE_Match object at 0x10069d370>
>>> pat.match(s.upper())                        # Doesn't match

>>> s.upper()                                    # Case folds

'STRASSE'
>>>

```

## 2.10.3

Unicode

<http://pypi.python.org/pypi/regex>

Unicode

## 2.11

### 2.11.1

## 2.11.2 strip()

`strip()` removes leading and trailing whitespace characters. `rstrip()` removes only trailing whitespace characters. `lstrip()` removes only leading whitespace characters.

```
>>> # Whitespace stripping
>>> s = ' hello world \n'
'
>>> s.strip()
'hello world'
>>> s.lstrip()
'hello world \n'
>>> s.rstrip()
' hello world'
>>>

>>> # Character stripping
>>> t = '-----hello====='
>>> t.lstrip('-')
'hello====='
>>> t.strip('-=')
'hello'
>>>
```

## 2.11.3 strip()

`strip()` removes leading and trailing whitespace characters. `rstrip()` removes only trailing whitespace characters. `lstrip()` removes only leading whitespace characters.

`strip()` removes leading and trailing whitespace characters. `rstrip()` removes only trailing whitespace characters. `lstrip()` removes only leading whitespace characters.

```
>>> s = ' hello      world      \n'
'
>>> s = s.strip()
>>> s
'hello      world'
>>>
```

replace() 方法

```
>>> s.replace(' ', '')
'helloworld'
>>> import re
>>> re.sub('\s+', ' ', s)
'hello world'
>>>
```

with open(filename) as f:  
lines = (line.strip() for line in f)  
for line in lines:  
 ...

```
with open(filename) as f:
    lines = (line.strip() for line in f)
    for line in lines:
        ...
```

lines = (line.strip() for line in f)  
lines[1]  
strip()

strip() 和 translate() 函数  
可以删除字符串中的指定字符

## 2.12 字符串操作

### 2.12.1 字符串

Web 浏览器“pýthöñ”  
显示为“python”

### 2.12.2 字符串

字符串操作函数包括 str.upper()、str.lower()、str.replace()、re.sub()、unicodedata.normalize() 等。  
str.upper() 和 str.lower() 将字符串中的所有字母转换为大写或小写。  
str.replace() 和 re.sub() 用于替换字符串中的子串。  
unicodedata.normalize() 用于将字符串转换为 Unicode 形式。

字符串操作函数包括 str.translate()。  
str.translate() 用于将字符串中的字符替换为其他字符。

```
>>> s = 'python\xf  
is\t'
```

```
awesome\r\n
,
>>> s
'pythøn\x0cis\tawesome\r\n'
>>>
```

translate()

```
>>> remap = {
...
ord('\t
') : ' ',
...
ord('\f
') : ' ',
...
ord('\r
') : None      # Deleted

...
}
>>> a = s.translate(remap)
>>> a
'pythøn is awesome\n'
>>>
```

\t \f  
\r

字典的键是Unicode码点，字典的值是Unicode码点

```
>>> import unicodedata
>>> import sys
>>> cmb_chrs = dict.fromkeys(c for c in range(sys.maxunicode)
...                           if
unicodedata.combining(chr(c)))
...
>>> b = unicodedata.normalize('NFD', a)
>>> b
'pyth n is awesome n'
>>> b.translate(cmb_chrs)
'python is awesome n'
>>>
```

字典的键是Unicode码点，字典的值是Unicode码点

字典的键是Unicode码点，字典的值是Unicode码点

字典的键是Unicode码点，字典的值是Unicode码点

```
>>> digitmap = { c: ord('0') + unicodedata.digit(chr(c))
...              for
c in
range(sys.maxunicode)
...              if
```

```

unicodedata.category(chr(c)) == 'Nd' }
...

>>> len(digitmap)
460
>>> # Arabic digits

>>> x = '\u0661\u0662\u0663'
'
>>> x.translate(digitmap)
'123'
>>>

```

0000000000000000I/O0000000000000000  
 00000000000000000000encode()decode()000000  
 00000000000000

```

>>> a
'python is awesome\n'
>>> b = unicodedata.normalize('NFD', a)
>>> b.encode('ascii', 'ignore').decode('ascii')
'python is awesome\n'
>>>

```

000normalize()00000000000000000000  
 ASCII00/00000000000000000000000000000000  
 00000000000000ASCII000000000000

## 2.12.3 00



字符串的replace()方法  
——字符串的replace()方法  
返回替换后的字符串

```
def  
clean_spaces(s):  
    s = s.replace('\r'  
, '')  
    s = s.replace('\t'  
, ' ')  
    s = s.replace('\f'  
, ' ')  
    return  
s
```

字符串的translate()方法  
——字符串的translate()方法

字符串的translate()方法  
——字符串的translate()方法

字符串的translate()方法  
——字符串的translate()方法  
“ ”

byte

## 2.13

### 2.13.1 ☐☐

## 2.13.2 ☐☐☐☐

```

    ljust() rjust()
    center()

```

```
>>> text = 'Hello World'
>>> text.ljust(20)
'Hello World  '
>>> text.rjust(20)
'      Hello World'
>>> text.center(20)
'      Hello World '
>>>
```

[illegible]

```
>>> text.rjust(20,'=')
'=====Hello World'
>>> text.center(20,'*')
'*****Hello World*****'
```

```
>>>
```

`format()` 函数可以指定文本的宽度，使其按指定的方式格式化。  
例如，使用 `<` 指定左对齐，使用 `>` 指定右对齐，使用 `^` 指定居中对齐。[\[2\]](#)

```
>>> format(text, '>20')
'          Hello World'
>>> format(text, '<20')
'Hello World '
>>> format(text, '^20')
'    Hello World    '
>>>
```

使用 `format()` 函数可以指定文本的宽度，使其按指定的方式格式化。

```
>>> format(text, '>=20s')
'====Hello World'
>>> format(text, '*^20s')
'****Hello World****'
>>>
```

使用 `format()` 函数可以指定文本的宽度，使其按指定的方式格式化。  
例如，使用 `{:10s}` 指定文本的宽度为 10 个字符，并左对齐。

```
>>> '{:>10s} {:>10s}'.format('Hello', 'World')
'      Hello      World'
>>>
```

`format()` 函数可以指定文本的宽度，使其按指定的方式格式化。  
例如，使用 `{:10s}` 指定文本的宽度为 10 个字符，并左对齐。

```
>>> x = 1.2345
>>> format(x, '>10')
'      1.2345'
>>> format(x, '^10.2f')
'    1.23  '
```

## 2.13.3 字符串格式化

字符串格式化的基本语法是：`%` 后面跟一个或多个格式说明符。

```
>>> '%-20s' % text
'Hello World '
>>> '%20s' % text
'      Hello World'
>>>
```

Python 提供了多种字符串格式化方法，包括 `format()`、`%` 运算符、`format()` 方法、`ljust()`、`rjust()`、`center()` 等。

关于 Python 字符串格式化的详细文档，请访问：  
<http://docs.python.org/3/library/string.html#formatspec>

## 2.14 列表和元组的格式化

## 2.14.1

## 2.14.2

```
>>> parts = ['Is', 'Chicago', 'Not', 'Chicago?']
>>> ' '.join(parts)
'Is Chicago Not Chicago?'
>>> ','.join(parts)
'Is,Chicago,Not,Chicago?'
>>> ''.join(parts)
'IsChicagoNotChicago?'
>>>
```

```
>>> a = 'Is Chicago'
>>> b = 'Not Chicago?'
>>> a + ' ' + b
```

```
'Is Chicago Not Chicago?'  
>>>
```

format() +

```
>>> print  
  
( '{} {}'.format(a,b)  
Is Chicago Not Chicago?  
>>> print  
  
(a + ' ' + b)  
Is Chicago Not Chicago?  
>>>
```

+

```
>>> a = 'Hello' 'World'  
>>> a  
'HelloWorld'  
>>>
```

## 2.14.3

字符串 + 字符串  
字符串 \* 整数  
字符串 \* 浮点数

```
s = ''  
for  
p in  
parts:  
    s += p
```

字符串.join() 字符串 + = 字符串  
字符串 \* 整数  
字符串 \* 浮点数

字符串 1.19  
字符串

```
>>> data = ['ACME', 50, 91.1]  
>>> ','.join(str(d) for  
d in  
data)  
'ACME,50,91.1'  
>>>
```

字符串 \* 整数  
字符串 \* 浮点数





yield 语句是 Python 中一个非常强大的工具，它允许你在一个函数中返回多个值，而不需要返回一个列表。yield 语句的语法如下：

```
def  
sample():  
    yield  
  
    'Is'  
    yield  
  
    'Chicago'  
    yield  
  
    'Not'  
    yield  
  
    'Chicago?'
```

join() 方法用于将列表中的元素连接成一个字符串。join() 方法的语法如下：

```
text = ''.join(sample())
```

I/O

```
for  
part in  
sample():
```

```
f.write(part)
```

□□□□□□□□□□□□□□I/O□□□□□□□□□□□□□□

```
def
combine(source, maxsize):
    parts = []
    size = 0
    for
part in
source:
    parts.append(part)
    size += len(part)
    if
size > maxsize:
        yield
''.join(parts)

parts = []

size = 0
    yield
''.join(parts)

for
part in
combine(sample(), 32768):
```

```
f.write(part)
```

## 2.15

### 2.15.1 ☐☐

## 2.15.2 ☐☐☐☐

Python `format()`

```
>>> s = '{name} has {n} messages.'
>>> s.format(name='Guido', n=37)
'Guido has 37 messages.'
>>>
```

```
format map() vars()
```

```
>>> name = 'Guido'
>>> n = 37
```

```
>>> s.format_map(vars())
'Guido has 37 messages.'
>>>
```

vars() returns a dictionary of an object's variables

```
>>> class Info:
...     def __init__(self, name, n):
...         self.name = name
...         self.n = n
...
>>> a = Info('Guido', 37)
>>> s.format_map(vars(a))
'Guido has 37 messages.'
>>>
```

format() format\_map() are useful for debugging  
if you want to print an object's variables

```
>>> s.format(name='Guido')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'n'
>>>
```

\_\_missing\_\_() is a method that can be defined in a dictionary subclass

```
class safesub(dict):
    def __missing__(self, key):
        return '{' + key + '}'
```

## format\_map() 函数

```
>>> del  
  
n      # Make sure n is undefined  
>>> s.format_map(safesub(vars()))  
'Guido has {n} messages.'  
>>>
```

format\_map() 函数使用字典作为参数，并返回格式化的字符串。它类似于 format() 函数，但使用字典而不是元组。以下是一个使用 format\_map() 函数的示例，它使用 sys.\_getframe() 函数来获取当前帧的局部变量字典，并将其传递给 safesub() 函数。这可以用于防止递归调用中的变量名冲突。

“frame  
hack” [3]

```
import sys  
  
def sub(text):  
    return text.format_map(safesub(sys._getframe(1).f_locals))
```

以下是一个使用 format\_map() 函数的示例，它使用 sys.\_getframe() 函数来获取当前帧的局部变量字典，并将其传递给 safesub() 函数。这可以用于防止递归调用中的变量名冲突。

```
>>> name = 'Guido'  
>>> n = 37  
>>> print  
  
(sub('Hello {name}'))  
Hello Guido  
>>> print  
  
(sub('You have {n} messages.'))  
You have 37 messages.  
>>> print  
  
(sub('Your favorite color is {color}'))  
Your favorite color is {color}
```

```
>>>
```

## 2.15.3 `%`

`%` is the old-style Python string formatting operator. It is used to format strings and is the most common way to format strings in Python. It is used by placing a percent sign (`%`) before the string to be formatted, followed by a colon (`:`) and a list of variables to be formatted. The variables are separated by commas. The format string can contain any valid Python expression, and the result is a string that is the result of the expression.

```
>>> name = 'Guido'
>>> n = 37
>>> '%(name) has %(n) messages.' % vars()
'Guido has 37 messages.'
>>>
```

`string.Template` is a class that provides a simple way to format strings. It is used by creating a `string.Template` object, passing it a string that contains the format string, and then calling the `substitute` method on the object, passing it a dictionary of variables. The result is a string that is the result of the substitution.

```
>>> import string
>>> s = string.Template('$name has $n messages.')
>>> s.substitute(vars())
'Guido has 37 messages.'
>>>
```

`format()` and `format_map()` are methods that provide a simple way to format strings. They are used by calling the method on a string, passing it a dictionary of variables. The result is a string that is the result of the formatting. `format()` is the preferred way to format strings, and `format_map()` is used when you want to format a string with a dictionary of variables.

```

    """
    __missing__(): Return the value of the safe_sub
    attribute.
    KeyError: If the attribute is not found.
    """
```

```

    sub() returns sys._getframe(1) if the
    attribute f_locals is not None.
    Otherwise, it returns the value of the
    attribute f_locals.
    """
    f_locals = getattr(self, 'f_locals', None)
    if f_locals is not None:
        return f_locals
    else:
        return sys._getframe(1)
```

## 2.16 2.16.1

### 2.16.1

```

    """
    """
```

### 2.16.2

```

    """textwrap"""
    """
```





## 2.16.3 文本

`textwrap` 模块提供了两个函数，`fill()` 和 `wrap()`，用于将长字符串分成多行。默认情况下，`fill()` 会在 80 个字符处换行。可以通过 `os.get_terminal_size()` 获取终端的宽度。

```
>>> import os
>>> os.get_terminal_size().columns
80
>>>
```

`fill()` 函数将长字符串分成多行，每行不超过指定的宽度。默认宽度为 80 个字符。可以通过 `os.get_terminal_size()` 获取终端的宽度。使用 `textwrap.TextWrapper` 类可以自定义换行规则。请参考 <http://docs.python.org/3.3/library/textwrap.html#textwrap.TextWrapper>。

## 2.17 使用 HTML 和 XML

### 2.17.1 转义

在 HTML 和 XML 文档中，某些字符需要转义。例如，小于号 `<` 和大于号 `>` 需要转义为 `&lt;` 和 `&gt;`。转义字符用于防止解析器误解文档结构。

### 2.17.2 转义

□□□□□□□□html.escape()□□□□□□□  
<or>□□□□□□□□□□□□□□□□□□□□

```
>>> s = 'Elements are written as "<tag>text</tag>".'
>>> import html

>>> print
(s)
Elements are written as "<tag>text</tag>".
>>> print
(html.escape(s))
Elements are written as "<tag>text</tag>".

>>> # Disable escaping of quotes

>>> print
(html.escape(s, quote=False))
Elements are written as "<tag>text</tag>".
>>>
```

□□□□□ASCII□□□□□□□□□ASCII□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□I/O□□□□□□□□□  
errors='xmlcharrefreplace'□□□□□□□□□□□□

```
>>> s = 'Spicy Jalapen~o'
>>> s.encode('ascii', errors='xmlcharrefreplace')
b'Spicy Jalape&#241;o'
>>>
```

HTML/XML escapes are used to represent characters that are not allowed in HTML/XML documents. For example, the less-than sign (<) is used to start a tag, so it must be escaped if you want to include it in the text of a document. The escape sequence for the less-than sign is <lt;.

HTML/XML escapes are used to represent characters that are not allowed in HTML/XML documents. For example, the less-than sign (<) is used to start a tag, so it must be escaped if you want to include it in the text of a document. The escape sequence for the less-than sign is <lt;.

```
>>> s = 'Spicy "Jalapeño&quot;.'
>>> from html.parser import

HTMLParser
>>> p = HTMLParser()
>>> p.unescape(s)
'Spicy "Jalapen~o".'
>>>

>>> t = 'The prompt is >>>'
>>> from xml.sax.saxutils import

unescape
>>> unescape(t)
'The prompt is >>>'
>>>
```

## 2.17.3 Escapes

HTML/XML escapes are used to represent characters that are not allowed in HTML/XML documents. For example, the less-than sign (<) is used to start a tag, so it must be escaped if you want to include it in the text of a document. The escape sequence for the less-than sign is <lt;. The `html.escape()` function escapes the less-than sign and other special characters.

HTML/XML 解析器  
xml.sax.saxutils.unescape()  
HTML/XML 解析器  
html.parser  
xml.etree.ElementTree

## 2.18 解析器

### 2.18.1 解析器

stream of tokens

### 2.18.2 解析器

```
text = 'foo = 23 + 42 * 10'
```

```
tokens = [('NAME', 'foo'), ('EQ', '='), ('NUM', '23'), ('PLUS', '+'),
```

```
('NUM', '42'), ('TIMES', '*'), ('NUM', 10)]
```

正则表达式匹配器

```
import re
NAME = r'(?P<NAME>[a-zA-Z_][a-zA-Z_0-9]*)'
NUM = r'(?P<NUM>\d+)'
PLUS = r'(?P<PLUS>\+)'
TIMES = r'(?P<TIMES>\*)'
EQ = r'(?P<EQ>=)'
WS = r'(?P<WS>\s+)'

master_pat = re.compile('|'.join([NAME, NUM, PLUS, TIMES, EQ,
WS]))
```

正则表达式匹配器?P<TOKENNAME>

正则表达式匹配器scanner()正则表达式匹配器match()正则表达式匹配器

```
>>> scanner = master_pat.scanner('foo = 42')
>>> scanner.match()
<_sre.SRE_Match object at 0x100677738>
>>> _.lastgroup, _.group()
('NAME', 'foo')
>>> scanner.match()
<_sre.SRE_Match object at 0x100677738>
>>> _.lastgroup, _.group()
('WS', ' ')
>>> scanner.match()
<_sre.SRE_Match object at 0x100677738>
```

```

>>> _.lastgroup, _.group()
('EQ', '=')
>>> scanner.match()
<_sre.SRE_Match object at 0x100677738>
>>> _.lastgroup, _.group()
('WS', ' ')
>>> scanner.match()
<_sre.SRE_Match object at 0x100677738>
>>> _.lastgroup, _.group()
('NUM', '42')
>>> scanner.match()
>>>

```

```

████████████████████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████████████████████

```

```

from collections import namedtuple

Token = namedtuple('Token', ['type', 'value'])
def generate_tokens(pat, text):
    scanner = pat.scanner(text)
    for m in iter(scanner.match, None):
        yield Token(m.lastgroup, m.group())

# Example use

for tok in generate_tokens(master_pat, 'foo = 42'):
    print(tok)

# Produces output

# Token(type='NAME', value='foo')

# Token(type='WS', value=' ')

# Token(type='EQ', value='=')

```

```
# Token(type='WS', value=' ')
```

```
# Token(type='NUM', value='42')
```

```
tokens = (tok for
tok in
generate_tokens(master_pat, text)
    if
tok.type != 'WS')
for
tok in
tokens:
    print
(tok)
```

### 2.18.3 ☐☐

正则表达式匹配字符串中的空白字符  
正则表达式匹配字符串中的空白字符WS

正则表达式匹配字符串中的空白字符  
re.compile('|'.join([NAME, NUM, PLUS, TIMES, EQ, WS]))正则表达式匹配字符串中的空白字符re  
正则表达式匹配字符串中的空白字符正则表达式匹配字符串中的空白字符  
正则表达式匹配字符串中的空白字符正则表达式匹配字符串中的空白字符

```
LT = r'(?P<LT><)'
LE = r'(?P<LE><=)'
EQ = r'(?P<EQ>=)'

master_pat = re.compile('|'.join([LE, LT, EQ])) # Correct

# master_pat = re.compile('|'.join([LT, LE, EQ])) # Incorrect
```

2正则表达式匹配字符串中的空白字符'<='  
正则表达式匹配字符串中的空白字符EQ='正则表达式匹配字符串中的空白字符LE  
正则表达式匹配字符串中的空白字符'<='正则表达式匹配字符串中的空白字符

正则表达式匹配字符串中的空白字符正则表达式匹配字符串中的空白字符  
正则表达式匹配字符串中的空白字符

```
PRINT = r'(P<PRINT>print)'
NAME = r'(P<NAME>[a-zA-Z_][a-zA-Z_0-9]*)'

master_pat = re.compile('|'.join([PRINT, NAME]))
```



```

for tok in generate_tokens(master_pat, 'printer'):
    print(tok)

# Outputs :

# Token(type='PRINT', value='print')

# Token(type='NAME', value='er')

```

PyParsing과PLY  
 PyParsing과PLY

## 2.19

### 2.19.1

### 2.19.2

BNF EBNF

```

expr ::= expr + term
      | expr - term
      | term
term  ::= term * factor
      | term / factor
      | factor
factor ::= ( expr )
        | NUM

```

□□□□EBNF□□□□□□□□□□

```

expr ::= term { (+|-) term }*
term  ::= factor { (*|/) factor }*
factor ::= ( expr )
         | NUM

```

□EBNF□□□□□□□{ ... }\*□□□□□□□□□\*□□□□  
 □□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□BNF□□□□□□□□□□□□□□□□□□□□□□□□  
 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□BNF□□□  
 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□3 + 4 \* 5□□□□  
 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□2.18□□□□□□□□  
 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

NUM + NUM \* NUM

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```

expr
expr ::= term { (+|-) term }*
expr ::= factor { (*|/) factor }* { (+|-) term }*
expr ::= NUM { (*|/) factor }* { (+|-) term }*
expr ::= NUM { (+|-) term }*
expr ::= NUM + term { (+|-) term }*
expr ::= NUM + factor { (*|/) factor }* { (+|-) term }*
expr ::= NUM + NUM { (*|/) factor }* { (+|-) term }*
expr ::= NUM + NUM * factor { (*|/) factor }* { (+|-) term }*
expr ::= NUM + NUM * NUM { (*|/) factor }* { (+|-) term }*
expr ::= NUM + NUM * NUM { (+|-) term }*
expr ::= NUM + NUM * NUM

```

NUM
 +
 { (\*|/) factor

NUM

```

import re
import collections

# Token specification

NUM      = r'(?P<NUM>\d+)'
PLUS     = r'(?P<PLUS>\+)'
MINUS    = r'(?P<MINUS>-)'
TIMES    = r'(?P<TIMES>\*)'
DIVIDE   = r'(?P<DIVIDE>/)'
LPAREN   = r'(?P<LPAREN>\(')
RPAREN   = r'(?P<RPAREN>\))'

```

```

WS = r'(?P<WS>\s+)'

master_pat = re.compile('|'.join([NUM, PLUS, MINUS, TIMES,
                                  DIVIDE, LPAREN, RPAREN,
WS]))
# Tokenizer

Token = collections.namedtuple('Token', ['type', 'value'])

def generate_tokens(text):
    scanner = master_pat.scanner(text)
    for m in iter(scanner.match, None):
        tok = Token(m.lastgroup, m.group())
        if tok.type != 'WS':
            yield tok

# Parser

class ExpressionEvaluator:
    '''

Implementation of a recursive descent parser. Each method

implements a single grammar rule. Use the ._accept() method

to test and accept the current lookahead token. Use the
    ._expect()

method to exactly match and discard the next token on on the
    input

(or raise a SyntaxError if it doesn't match).

    '''

```

```

def parse(self, text):
    self.tokens = generate_tokens(text)
    self.tok = None          # Last symbol consumed

    self.nexttok = None      # Next symbol tokenized

    self._advance()          # Load first lookahead token

    return self.expr()

def _advance(self):
    'Advance one token ahead'
    self.tok, self.nexttok = self.nexttok, next(self.tokens,
None)

def _accept(self, toktype):
    'Test and consume the next token if it matches toktype'
    if self.nexttok and self.nexttok.type == toktype:
        self._advance()
        return True
    else:
        return False

def _expect(self, toktype):
    'Consume next token if it matches toktype or raise
SyntaxError'
    if not self._accept(toktype):
        raise SyntaxError('Expected ' + toktype)

# Grammar rules follow

def expr(self):
    "expression ::= term { ('+'|'-') term }*"

    exprval = self.term()
    while self._accept('PLUS') or self._accept('MINUS'):
        op = self.tok.type
        right = self.term()
        if op == 'PLUS':
            exprval += right

```

```

        elif op == 'MINUS':
            exprval -= right
        return exprval

def term(self):
    "term ::= factor { ('*' | '/') factor }*"

    termval = self.factor()
    while self._accept('TIMES') or self._accept('DIVIDE'):
        op = self.tok.type
        right = self.factor()
        if op == 'TIMES':
            termval *= right
        elif op == 'DIVIDE':
            termval /= right
    return termval

def factor(self):
    "factor ::= NUM | ( expr )"

    if self._accept('NUM'):
        return int(self.tok.value)
    elif self._accept('LPAREN'):
        exprval = self.expr()
        self._expect('RPAREN')
        return exprval
    else:
        raise SyntaxError('Expected NUMBER or LPAREN')

```

□□□□□□□□□□
ExpressionEvaluator
□□□

□□

```

>>> e = ExpressionEvaluator()
>>> e.parse('2')
2
>>> e.parse('2 + 3')
5
>>> e.parse('2 + 3 * 4')
14
>>> e.parse('2 + (3 + 4) * 5')

```



```

        exprval = ('-', exprval, right)
    return exprval

def term(self):
    "term ::= factor { ('*' | '/') factor }"

    termval = self.factor()
    while self._accept('TIMES') or self._accept('DIVIDE'):
        op = self.tok.type
        right = self.factor()
        if op == 'TIMES':
            termval = ('*', termval, right)
        elif op == 'DIVIDE':
            termval = ('/', termval, right)
    return termval

def factor(self):
    "factor ::= NUM | ( expr )"

    if self._accept('NUM'):
        return int(self.tok.value)
    elif self._accept('LPAREN'):
        exprval = self.expr()
        self._expect('RPAREN')
        return exprval
    else:
        raise SyntaxError('Expected NUMBER or LPAREN')

```

□□□□□□□□□□□□□□□□

```

>>> e = ExpressionTreeBuilder()
>>> e.parse('2 + 3')
('+', 2, 3)
>>> e.parse('2 + 3 * 4')
('+', 2, ('*', 3, 4))
>>> e.parse('2 + (3 + 4) * 5')
('+', 2, ('*', ('+', 3, 4), 5))
>>> e.parse('2 + 3 + 4')
('+', ('+', 2, 3), 4)
>>>

```



## 2.19.3 ☐☐

The diagram consists of three horizontal rows of boxes. The top row contains 15 boxes, the middle row contains 20 boxes, and the bottom row contains 15 boxes. The boxes are arranged in a staggered pattern, with the middle row starting from the left edge and the top and bottom rows starting from the center of the middle row's span.

```

expr ::= term { ('+' | '-') term }*
term  ::= factor { ('*' | '/') factor }*
factor ::= '(' expr ')'
        | NUM

```

```
class ExpressionEvaluator:
    ...
    def expr(self):
    ...
    def term(self):
    ...
    def factor(self):
    ...
```

- term → factor — factor ::= '(' expr ')'

expr → factor —
- expr → factor — expect()
- expr → factor — accept() — except() — accept()
- expr ::= term { ('+' | '-') term }\*, while
- expr ::= term { ('+' | '-') term }\*, while

Python 的语法是  
Python 的语法是

Python 的语法是  
Python 的语法是  
Grammar/Grammar

Python 的语法是  
Python 的语法是

```
items ::= items ',' item  
        | item
```

items()

```
def  
items(self):  
    itemsval = self.items()  
    if  
itemsval and  
self._accept(','):   
        itemsval.append(self.item())  
    else  
:  
    itemsval = [ self.item() ]
```

[illegible]

```
from ply.lex import lex
from ply.yacc import yacc

# Token list

tokens = [ 'NUM', 'PLUS', 'MINUS', 'TIMES', 'DIVIDE',
           'LPAREN', 'RPAREN' ]

# Ignored characters

t_ignore = ' \t\n'

# Token specifications (as regexs)
```

```

t_PLUS = r'\+'
t_MINUS = r'\-'
t_TIMES = r'\*'
t_DIVIDE = r'\/'
t_LPAREN = r'\('
t_RPAREN = r'\)'

# Token processing functions

def t_NUM(t):
    r'\d+'
    t.value = int(t.value)
    return t

# Error handler

def t_error(t):
    print('Bad character: {!r}'.format(t.value[0]))
    t.skip(1)

# Build the lexer

lexer = lex()

# Grammar rules and handler functions

def p_expr(p):
    '''
        expr : expr PLUS term

        / expr MINUS term

        ...
    '''

```

```

if p[2] == '+':
    p[0] = p[1] + p[3]
elif p[2] == '-':
    p[0] = p[1] - p[3]

def p_expr_term(p):
    '''

    expr : term

    ...

    p[0] = p[1]

def p_term(p):
    '''

    term : term TIMES factor

    / term DIVIDE factor

    ...

    if p[2] == '*':
        p[0] = p[1] * p[3]
    elif p[2] == '/':
        p[0] = p[1] / p[3]

def p_term_factor(p):
    '''

    term : factor

    ...

```

```

    p[0] = p[1]
def p_factor(p):
    '''

    factor : NUM

    '''

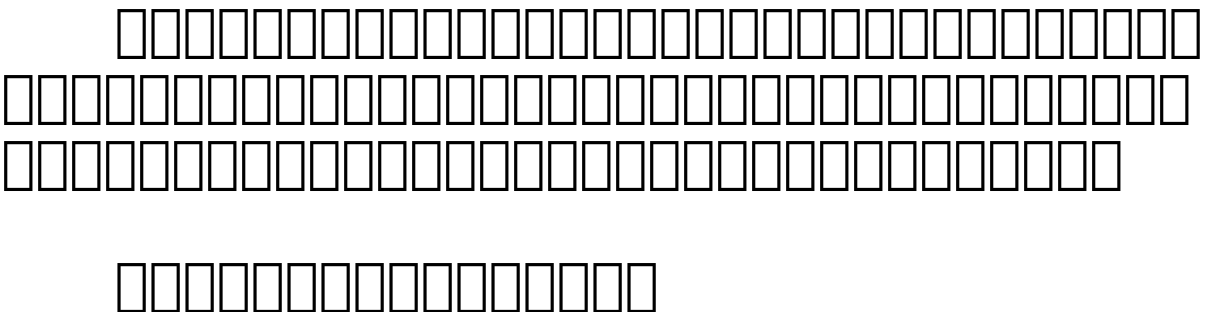
    p[0] = p[1]
def p_factor_group(p):
    '''

    factor : LPAREN expr RPAREN

    '''

    p[0] = p[2]
def p_error(p):
    print('Syntax error')
parser = yacc()

```



```
>>> parser.parse('2')
2
>>> parser.parse('2+3')
5
>>> parser.parse('2+(3+4)*5')
37
>>>
```

Python의 ast 모듈은 Python 코드를 추상 구문 트리(abstract syntax tree)로 변환하는 데 사용됩니다. 이 모듈은 Python의 파싱 결과를 표현하는 데 사용되며, 코드의 구조를 분석하고 변환하는 데 유용합니다.

## 2.20 문자열과 바이트

### 2.20.1 문자열

Python에서는 문자열을 `str` 타입으로 표현하며, 바이트 문자열은 `bytes` 타입으로 표현합니다. `Byte String`은 바이트 문자열을 가리키는 용어입니다.

### 2.20.2 바이트 문자열

바이트 문자열은 `bytes` 타입으로 표현되며, 문자열과 유사한 방식으로 사용됩니다.

```
>>> data = b'Hello World'
>>> data[0:5]
b'Hello'
>>> data.startswith(b'Hello')
True
>>> data.split()
```



```
[b'Hello', b'World']
>>> data.replace(b'Hello', b'Hello Cruel')
b'Hello Cruel World'
>>>
```

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

```
>>> data = bytearray(b'Hello World')
>>> data[0:5]
bytearray(b'Hello')
>>> data.startswith(b'Hello')
True
>>> data.split()
[bytearray(b'Hello'), bytearray(b'World')]
>>> data.replace(b'Hello', b'Hello Cruel')
bytearray(b'Hello Cruel World')
>>>
```

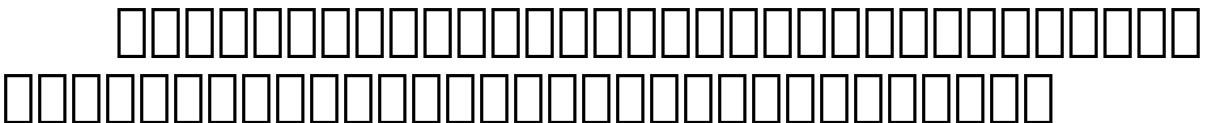
```
>>>
>>> data = b'FOO:BAR,SPAM'
>>> import re

>>> re.split('[:,]',data)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    File "/usr/local/lib/python3.3/re.py", line 191, in split
      return
    _compile(pattern, flags).split(string, maxsplit)
TypeError: can't use a string pattern on a bytes-like object

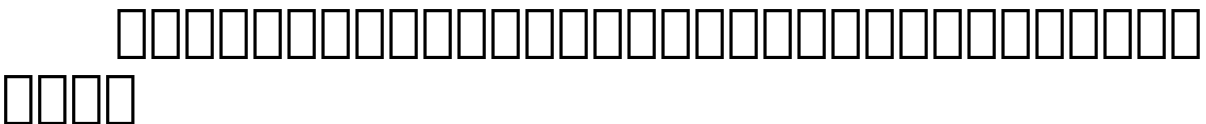
>>> re.split(b'[:,]',data)  # Notice: pattern as bytes
```

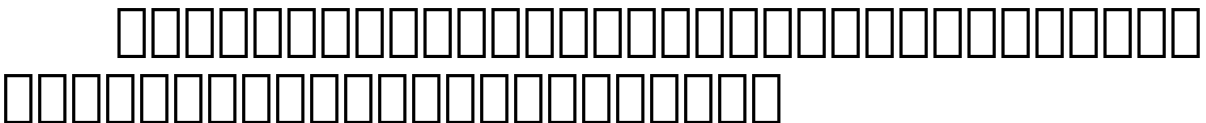
```
[b'F00', b'BAR', b'SPAM']  
>>>
```

## 2.20.3



```
>>> a = 'Hello World'      # Text string  
  
>>> a[0]  
'H'  
>>> a[1]  
'e'  
>>> b = b'Hello World'    # Byte string  
  
>>> b[0]  
72  
>>> b[1]  
101  
>>>
```





```
>>> s = b'Hello World'  
>>> print
```

```
(s)
b'Hello World' # Observe b'...'
>>> print

(s.decode('ascii'))
Hello World
>>>
```

[illegible]

```
>>> b'%10s %10d %10.2f' % (b'ACME', 100, 490.1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for %: 'bytes' and
'tuple'

>>> b'{} {} {}'.format(b'ACME', 100, 490.1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'bytes' object has no attribute 'format'
>>>
```

```
>>> '{:10s} {:10d} {:10.2f}'.format('ACME', 100,
490.1).encode('ascii')
b'ACME                100          490.10'
>>>
```

[illegible]

```

>>> # Write a UTF-8 filename
>>> with
open('jalape\xfl
o.txt', 'w') as
f:
...
f.write('spicy')
...

>>> # Get a directory listing
>>> import os


>>> os.listdir('.')          # Text string (names are
decoded)
['Jalapeño.txt']
>>> os.listdir(b'.')        # Byte string (names left as
bytes)
[b'jalapen\xcc\x83o.txt']
>>>

```

UTF-8 5.15

Unicode Python

[illegible]

[1] 

[2]    '>'<>'<'<'^'<  
———

```
[3] In sys._getframe():
      1: ~~~~~
```

## 3 四舍五入

Python 四舍五入的函数是 `round()`。它接受两个参数：要四舍五入的数值和要保留的小数位数。例如，`round(1.23, 1)` 将 1.23 四舍五入到一位小数，结果为 1.2。

### 3.1 四舍五入

#### 3.1.1 四舍五入

四舍五入的函数是 `round()`。

#### 3.1.2 四舍五入

四舍五入的函数是 `round(value, ndigits)`，其中 `value` 是要四舍五入的数值，`ndigits` 是要保留的小数位数。

```
>>> round(1.23, 1)
1.2
>>> round(1.27, 1)
1.3
>>> round(-1.27, 1)
-1.3
>>> round(1.25361, 3)
1.254
>>>
```

round() 函数可以按指定的精度对浮点数进行四舍五入。  
例如，将 1.5 四舍五入到最近的整数，结果为 2。

round() 函数还可以指定要保留的小数位数。  
例如，将 1.23456 四舍五入到小数点后两位，结果为 1.23。

```
>>> a = 1627731
>>> round(a, -1)
1627730
>>> round(a, -2)
1627700
>>> round(a, -3)
1628000
>>>
```

### 3.1.3 字符串格式化

字符串格式化是指将变量或表达式嵌入到字符串中，以生成格式化的输出。  
Python 提供了多种字符串格式化方法，其中最常用的是 format() 函数。  
例如，将变量 x 的值格式化为字符串，并嵌入到另一个字符串中。

```
>>> x = 1.23456
>>> format(x, '0.2f')
'1.23'
>>> format(x, '0.3f')
'1.235'
>>> 'value is {:.3f}'.format(x)
'value is 1.235'
>>>
```

Python 2.7 版本中，浮点数的精度问题是一个常见的陷阱。在 Python 3 中，浮点数的精度问题依然存在，但处理方式有所不同。

```
>>> a = 2.1
>>> b = 4.2
>>> c = a + b
>>> c
6.300000000000001
>>> c = round(c, 2)      # "Fix" result (???)

>>> c
6.3
>>>
```

在 Python 中，浮点数的精度问题是一个常见的陷阱。在 Python 3 中，浮点数的精度问题依然存在，但处理方式有所不同。decimal 模块提供了一种高精度的浮点数表示方法。

## 3.2 浮点数的精度问题

### 3.2.1 精度问题

在 Python 中，浮点数的精度问题是一个常见的陷阱。在 Python 3 中，浮点数的精度问题依然存在，但处理方式有所不同。

### 3.2.2 精度问题



Python float 数据类型使用 IEEE 754 标准  
表示浮点数，这导致了一些精度问题。

```
>>> a = 4.2
>>> b = 2.1
>>> a + b
6.300000000000001
>>> (a + b) == 6.3
False
>>>
```

Python 的 float 数据类型使用 IEEE 754 标准  
表示浮点数，这导致了一些精度问题。  
Python 的 float 数据类型使用 IEEE 754 标准  
表示浮点数，这导致了一些精度问题。

decimal 模块

```
>>> from decimal import Decimal
>>> a = Decimal('4.2')
>>> b = Decimal('2.1')
>>> a + b
Decimal('6.3')
>>> print(a + b)
6.3
>>> (a + b) == Decimal('6.3')
True
>>>
```

Decimal 数据类型使用十进制表示浮点数，  
避免了精度问题。

decimal模块提供了Decimal类，用于表示十进制数。它还可以提供上下文管理器，用于控制精度和舍入模式。

decimal模块提供了Decimal类，用于表示十进制数。它还可以提供上下文管理器，用于控制精度和舍入模式。

```
>>> from decimal import localcontext
>>> a = Decimal('1.3')
>>> b = Decimal('1.7')
>>> print(a / b)
0.7647058823529411764705882353
>>> with localcontext() as ctx:
...     ctx.prec = 3
...     print(a / b)
...
0.765
>>> with localcontext() as ctx:
...     ctx.prec = 50
...     print(a / b)
...
0.76470588235294117647058823529411764705882352941176
>>>
```

### 3.2.3 十进制

decimal模块提供了Decimal类，用于表示十进制数。它还可以提供上下文管理器，用于控制精度和舍入模式。

Python提供了decimal模块，用于表示十进制数。它还可以提供上下文管理器，用于控制精度和舍入模式。

17 float 17  
——

subtractive cancellation

```
>>> nums = [1.23e+18, 1, -1.23e+18]
>>> sum(nums)      # Notice how 1 disappears

0.0
>>>
```

math.fsum()

```
>>> import math
>>> math.fsum(nums)
1.0
>>>
```

error propagation

decimal 模块提供高精度十进制浮点数的支持。decimal 模块是 Python 标准库的一部分，它提供了 Decimal 类——一个高精度的十进制浮点数类。

## 3.3 十进制浮点数

### 3.3.1 安装

decimal 模块是 Python 标准库的一部分，因此不需要单独安装。它位于 Python 的 `decimal` 包中。

### 3.3.2 使用

decimal 模块提供了 `format()` 方法，用于格式化十进制浮点数。

```
>>> x = 1234.56789
>>> # Two decimal places of accuracy
>>> format(x, '0.2f')
'1234.57'
>>> # Right justified in 10 chars, one-digit accuracy
>>> format(x, '>10.1f')
```

```
' 1234.6'

>>> # Left justified

>>> format(x, '<10.1f')
'1234.6 '

>>> # Centered

>>> format(x, '^10.1f')
' 1234.6 '

>>> # Inclusion of thousands separator

>>> format(x, ',')
'1,234.56789'
>>> format(x, '0,.1f')
'1,234.6'
>>>
```

float to string: `format(x, 'E')`  
 string to float: `float(s)`

```
>>> format(x, 'e')
'1.234568e+03'
>>> format(x, '0.2E')
'1.23E+03'
>>>
```

String formatting: `'[<>^]?'`  
`width[,]?(.digits)?'widthdigits'`  
`format()`

```
>>> 'The value is {:0,.2f}'.format(x)
'The value is 1,234.57'
>>>
```

### 3.3.3 四舍五入

Python 3 中，`round()` 函数用于对浮点数进行四舍五入。它接受两个参数：要四舍五入的浮点数和要保留的小数位数。返回值为四舍五入后的浮点数。

例如，将 1234.56789 四舍五入到小数点后一位，可以使用 `round(1234.56789, 1)` 实现。

```
>>> x
1234.56789
>>> format(x, '0.1f')
'1234.6'
>>> format(-x, '0.1f')
'-1234.6'
>>>
```

在 Python 3 中，`format()` 函数可以用于格式化字符串。它接受两个参数：要格式化的对象和格式字符串。返回值为格式化后的字符串。

```
>>> swap_separators = { ord('.'): ',', ord(','): '.' }
>>> format(x, ',').translate(swap_separators)
'1.234,56789'
>>>
```

Python 的字符串格式化操作符

```
>>> '%0.2f' % x
'1234.57'
>>> '%10.1f' % x
'    1234.6'
>>> '%-10.1f' % x
'1234.6 '
```

Python 的字符串格式化操作符 `format()` 是 Python 3 中引入的，它比 `%` 操作符更强大，也更安全。

## 3.4 字符串格式化

### 3.4.1 字符串

字符串格式化操作符

### 3.4.2 字符串

字符串格式化操作符 `bin()` `oct()` `hex()` 用于将数字转换为二进制、八进制和十六进制的字符串表示。

```
>>> x = 1234
>>> bin(x)
'0b10011010010'
```

```
>>> oct(x)
'0o2322'
>>> hex(x)
'0x4d2'
>>>
```

```
format('0b%0o%0x') % 10
```

```
>>> format(x, 'b')
'10011010010'
>>> format(x, 'o')
'2322'
>>> format(x, 'x')
'4d2'
>>>
```

```
>>> x = -1234
>>> format(x, 'b')
'-10011010010'
>>> format(x, 'x')
'-4d2'
>>>
```

**32**

```
>>> x = -1234
>>> format(2**32 + x, 'b')
'11111111111111111111111111111111011001011110'
```



```
>>> format(2**32 + x, 'x')
'fffffb2e'
>>>
```

```
int()
```

```
>>> int('4d2', 16)
1234
>>> int('10011010010', 2)
1234
>>>
```

### 3.4.3 〇〇

The diagram consists of three rows of boxes. The top row has 20 boxes, the middle row has 20 boxes, and the bottom row has 5 boxes. The boxes are arranged in a staggered pattern, with the top row starting at column 4, the middle row starting at column 1, and the bottom row starting at column 1.

Python

```
>>> import os  
>>> os.chmod('script.py', 0755)  
File "<stdin>", line 1  
    os.chmod('script.py', 0755)  
                        ^  
SyntaxError: invalid token  
>>>
```

0o

```
>>> os.chmod('script.py', 0o755)
>>>
```

## 3.5

### 3.5.1

### 3.5.2

16 128

```
data = b'\x00\x124V\x00x\x90\xab\x00\xcd\xef\x01\x00#\x004'
```

`int.from_bytes()`

```
>>> len(data)
16
>>> int.from_bytes(data, 'little')
69120565665751139577663547927094891008
>>> int.from_bytes(data, 'big')
```

```
94522842520747284487117727783387188
>>>
```

int.to\_bytes()  
16

```
>>> x = 94522842520747284487117727783387188
>>> x.to_bytes(16, 'big')
b'\x00\x124V\x00x\x90\xab\x00\xcd\xef\x01\x00#\x004'
>>> x.to_bytes(16, 'little')
b'4\x00#\x00\x01\xef\xcd\x00\xab\x90x\x00V4\x12\x00'
>>>
```

### 3.5.3

IPv6  
128

struct  
6.11 struct

```
>>> data
b'\x00\x124V\x00x\x90\xab\x00\xcd\xef\x01\x00#\x004'
>>> import struct
```

```
>>> hi, lo = struct.unpack('>QQ', data)
>>> (hi << 64) + lo
94522842520747284487117727783387188
>>>
```

```
>>> x = 0x01020304
>>> x.to_bytes(4, 'big')
b'\x01\x02\x03\x04'
>>> x.to_bytes(4, 'little')
b'\x04\x03\x02\x01'
>>>
```

```
int.bit_length()
```

```
>>> x = 523 ** 23
>>> x
33538130011366187510753685271401905616035565533397884901794406
7
>>> x.to_bytes(16, 'little')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
OverflowError: int too big to convert
>>> x.bit_length()
208
>>> nbytes, rem = divmod(x.bit_length(), 8)
>>> if
rem:
...
```

```

nbytes += 1
...

>>>
>>> x.to_bytes(nbytes, 'little')
b'\x03X\xf1\x82iT\x96\xac\xc7c\x16\xf3\xb9\xcf...\xd0'
>>>

```

## 3.6 字符串

### 3.6.1 字符串

字符串是Python中一种不可变的数据类型，用于存储文本信息。字符串可以包含字母、数字、符号以及换行符等。字符串的索引从0开始，可以通过索引访问字符串中的单个字符。字符串还支持切片操作，可以提取字符串的一部分。此外，字符串还可以与数字进行拼接，形成更复杂的文本信息。

### 3.6.2 复数

复数是Python中一种特殊的数据类型，用于表示复数。复数由实部和虚部组成，通常表示为  $a + bj$  的形式，其中  $a$  是实部， $b$  是虚部， $j$  是虚数单位。复数可以进行加、减、乘、除等运算。Python提供了 `complex()` 函数来创建复数，以及 `real` 和 `imag` 属性来访问复数的实部和虚部。

```

>>> a = complex(2, 4)
>>> b = 3 - 5j
>>> a
(2+4j)
>>> b
(3-5j)
>>>

```

□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> a.real
2.0
>>> a.imag
4.0
>>> a.conjugate()
(2-4j)
>>>
```

□□□□□□□□□□□□□□□□□□□□□□

```
>>> a + b
(5-1j)
>>> a * b
(26+2j)
>>> a / b
(-0.4117647058823529+0.6470588235294118j)
>>> abs(a)
4.47213595499958
>>>
```

□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□cmath□□□

```
>>> import cmath
>>> cmath.sin(a)
(24.83130584894638-11.356612711218174j)
>>> cmath.cos(a)
(-11.36423470640106-24.814651485634187j)
>>> cmath.exp(a)
(-4.829809383269385-5.5920560936409816j)
>>>
```

### 3.6.3 `np`

Python에서 `numpy` 모듈을 사용하기 위해서는 `import numpy as np`를 먼저 실행해야 합니다.

```
>>> import numpy as np
>>> a = np.array([2+3j, 4+5j, 6-7j, 8+9j])
>>> a
array([ 2.+3.j, 4.+5.j, 6.-7.j, 8.+9.j])
>>> a + 2
array([ 4.+3.j, 6.+5.j, 8.-7.j, 10.+9.j])
>>> np.sin(a)
array([ 9.15449915 -4.16890696j, -56.16227422 -48.50245524j,
       -153.20827755-526.47684926j, 4008.42651446-
       589.49948373j])
>>>
```

Python에서 `math` 모듈을 사용하기 위해서는 `import math`를 먼저 실행해야 합니다.

```
>>> import math
>>> math.sqrt(-1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: math domain error
>>>
```

Python에서 `cmath` 모듈을 사용하기 위해서는 `import cmath`를 먼저 실행해야 합니다.

```
>>> import cmath
>>> cmath.sqrt(-1)
1j
```

```
>>>
```

## 3.7 浮点数NaN

### 3.7.1 简介

NaN (Not a Number) 不是一个数字

### 3.7.2 浮点数

Python 使用 `float()` 函数来创建浮点数

```
>>> a = float('inf')
>>> b = float('-inf')
>>> c = float('nan')
>>> a
inf
>>> b
-inf
>>> c
nan
>>>
```

使用 `math.isinf()` 和 `math.isnan()` 来检查浮点数



```
>>> math.isinf(a)
True
>>> math.isnan(c)
True
>>>
```

### 3.7.3 浮点

浮点数的存储格式遵循IEEE 754标准。浮点数的存储格式如下：

浮点数的存储格式如下：

```
>>> a = float('inf')
>>> a + 45
inf
>>> a * 10
inf
>>> 10 / a
0.0
>>>
```

浮点数的存储格式如下：NaN（Not a Number）

```
>>> a = float('inf')
>>> a/a
nan
>>> b = float('-inf')
>>> a + b
nan
>>>
```

## NaN 浮点非数值

```
>>> c = float('nan')
>>> c + 23
nan
>>> c / 2
nan
>>> c * 2
nan
>>> math.sqrt(c)
nan
>>>
```

NaN 浮点非数值

NaN

```
>>> c = float('nan')
>>> d = float('nan')
>>> c == d
False
>>> c is
d
False
>>>
```

NaN 浮点非数值

math.isnan() 浮点非数值

NaN 浮点非数值 Python

fpectl 浮点非数值

Python 浮点非数值

Python 浮点非数值

<http://docs.python.org/3/library/fractions.html>

## 3.8 fractions

### 3.8.1 Fractions

The fractions module implements rational numbers as objects. It provides a Fraction class that can be used to create and manipulate fractions. Fractions are immutable and hashable, and can be used as dictionary keys. Fractions are also comparable, and can be ordered. Fractions are also iterable, and can be converted to and from strings.

### 3.8.2 Fractions

The fractions module provides a Fraction class that can be used to create and manipulate fractions.

```
>>> from fractions import Fraction
>>> a = Fraction(5, 4)
>>> b = Fraction(7, 16)
>>> print(a + b)
27/16
>>> print(a * b)
35/64

>>> # Getting numerator/denominator

>>> c = a * b
>>> c.numerator
35
>>> c.denominator
64
>>> # Converting to a float
```

```

>>> float(c)
0.546875

>>> # Limiting the denominator of a value

>>> print(c.limit_denominator(8))
4/7

>>> # Converting a float to a fraction

>>> x = 3.75
>>> y = Fraction(*x.as_integer_ratio())
>>> y
Fraction(15, 4)
>>>

```

### 3.8.3 `Decimal`

`Decimal` 클래스는 고정 소수점 산술을 위한 클래스입니다. 이 클래스는 `float` 클래스와 유사하지만, `float` 클래스는 이진 소수점 산술을 사용하며, `Decimal` 클래스는 십진 소수점 산술을 사용합니다.

## 3.9 `Grid`

### 3.9.1 `Grid`

`Grid` 클래스는 2D 그리드를 나타내는 클래스입니다. 이 클래스는 `grid` 속성을 가지며, 이 속성은 그리드의 상태를 나타냅니다.

### 3.9.2 `Grid`

NumPy  
NumPy Python Python  
NumPy

```
>>> # Python lists
>>> x = [1, 2, 3, 4]
>>> y = [5, 6, 7, 8]
>>> x * 2
[1, 2, 3, 4, 1, 2, 3, 4]
>>> x + 10
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate list (not "int") to list
>>> x + y
[1, 2, 3, 4, 5, 6, 7, 8]

>>> # Numpy arrays
>>> import numpy as np

>>> ax = np.array([1, 2, 3, 4])
>>> ay = np.array([5, 6, 7, 8])
>>> ax * 2
array([2, 4, 6, 8])
>>> ax + 10
array([11, 12, 13, 14])
>>> ax + ay
array([ 6,  8, 10, 12])
>>> ax * ay
array([ 5, 12, 21, 32])
>>>
```

NumPy  $ax * 2$   $ax + 10$   
NumPy

NumPy 是 Python 中用于科学计算的库，它提供了高效的数组对象和大量的数学函数。NumPy 的数组对象是 Python 列表的超集，但它的操作效率要高得多。

NumPy 的数组对象是 Python 列表的超集，但它的操作效率要高得多。NumPy 的数组对象是 Python 列表的超集，但它的操作效率要高得多。

```
>>> def f(x):  
...     return 3*x**2 - 2*x + 7  
...  
>>> f(ax)  
array([ 8, 15, 28, 47])  
>>>
```

NumPy 提供了大量的数学函数，包括三角函数、指数函数、对数函数等。这些函数可以直接应用于 NumPy 数组，而不需要编写循环。

```
>>> np.sqrt(ax)  
array([ 1.          ,  1.41421356,  1.73205081,  2.        ])  
>>> np.cos(ax)  
array([ 0.54030231, -0.41614684, -0.9899925 , -0.65364362])  
>>>
```

NumPy 的数组对象是 Python 列表的超集，但它的操作效率要高得多。NumPy 的数组对象是 Python 列表的超集，但它的操作效率要高得多。

NumPy 的数组对象是 Python 列表的超集，但它的操作效率要高得多。NumPy 的数组对象是 Python 列表的超集，但它的操作效率要高得多。

# NumPy Python 10000×10000

```
>>> grid = np.zeros(shape=(10000,10000), dtype=float)
>>> grid
array([[ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       ...,
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.]])
>>>
```

```
>>> grid += 10
>>> grid
array([[ 10.,  10.,  10., ...,  10.,  10.,  10.],
       [ 10.,  10.,  10., ...,  10.,  10.,  10.],
       [ 10.,  10.,  10., ...,  10.,  10.,  10.],
       ...,
       [ 10.,  10.,  10., ...,  10.,  10.,  10.],
       [ 10.,  10.,  10., ...,  10.,  10.,  10.],
       [ 10.,  10.,  10., ...,  10.,  10.,  10.]])
>>> np.sin(grid)
array([[ -0.54402111, -0.54402111, -0.54402111, ...,
        -0.54402111,
        -0.54402111, -0.54402111],
       [ -0.54402111, -0.54402111, -0.54402111, ...,
        -0.54402111,
        -0.54402111, -0.54402111],
       [ -0.54402111, -0.54402111, -0.54402111, ...,
        -0.54402111,
        -0.54402111, -0.54402111],
       ...,
       [ -0.54402111, -0.54402111, -0.54402111, ...,
        -0.54402111,
        -0.54402111, -0.54402111],
       [ -0.54402111, -0.54402111, -0.54402111, ...,
        -0.54402111,
        -0.54402111, -0.54402111],
       [ -0.54402111, -0.54402111, -0.54402111, ...,
        -0.54402111,
        -0.54402111, -0.54402111]])
```

```

        [-0.54402111, -0.54402111, -0.54402111, ...,
        -0.54402111,
         -0.54402111, -0.54402111],
        [-0.54402111, -0.54402111, -0.54402111, ...,
        -0.54402111,
         -0.54402111, -0.54402111]])
>>>

```

NumPy NumPy  
 Python — NumPy  
 Python — NumPy

```

>>> a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11,
12]])
>>> a
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])

>>> # Select row 1

>>> a[1]
array([5, 6, 7, 8])

>>> # Select column 1

>>> a[:,1]
array([ 2,  6, 10])

>>> # Select a subregion and change it

>>> a[1:3, 1:3]
array([[ 6,  7],
       [10, 11]])
>>> a[1:3, 1:3] += 10
>>> a
array([[ 1,  2,  3,  4],

```



```

        [ 5, 16, 17, 8],
        [ 9, 20, 21, 12]])

>>> # Broadcast a row vector across an operation on all rows

>>> a + [100, 101, 102, 103]
array([[101, 103, 105, 107],
       [105, 117, 119, 111],
       [109, 121, 123, 115]])
>>> a
array([[ 1,  2,  3,  4],
       [ 5, 16, 17,  8],
       [ 9, 20, 21, 12]])

>>> # Conditional assignment on an array

>>> np.where(a < 10, a, 10)
array([[ 1,  2,  3,  4],
       [ 5, 10, 10,  8],
       [ 9, 10, 10, 10]])
>>>

```

### 3.9.3 安装

Python 安装 NumPy 的步骤如下：

1. 安装 NumPy 的依赖库（如 C 编译器、BLAS、LAPACK 等）。

2. 安装 NumPy 本身。

```

import numpy as np

```

NumPy  
<http://www.numpy.org>

## 3.10

### 3.10.1

### 3.10.2

NumPy `matrix`  
Matrix 3.9

```
>>> import numpy as np
>>> m = np.matrix([[1,-2,3],[0,4,5],[7,8,-9]])
>>> m
matrix([[ 1, -2,  3],
        [ 0,  4,  5],
        [ 7,  8, -9]])

>>> # Return transpose

>>> m.T
matrix([[ 1,  0,  7],
        [-2,  4,  8],
        [ 3,  5, -9]])

>>> # Return inverse
```

```

>>> m.I
matrix([[ 0.33043478, -0.02608696, 0.09565217],
        [-0.15217391, 0.13043478, 0.02173913],
        [ 0.12173913, 0.09565217, -0.0173913 ]])

>>> # Create a vector and multiply

>>> v = np.matrix([[2],[3],[4]])
>>> v
matrix([[2],
        [3],
        [4]])
>>> m * v
matrix([[ 8],
        [32],
        [ 2]])
>>>

```

## numpy.linalg

```

>>> import numpy.linalg

>>> # Determinant

>>> numpy.linalg.det(m)
-229.99999999999983

>>> # Eigenvalues

>>> numpy.linalg.eigvals(m)
array([-13.11474312,  2.75956154,  6.35518158])

>>> # Solve for x in mx = v

>>> x = numpy.linalg.solve(m, v)
>>> x

```

```
matrix([[ 0.96521739],
        [ 0.17391304],
        [ 0.46086957]])
>>> m * x
matrix([[ 2.],
        [ 3.],
        [ 4.]])
>>> v
matrix([[2],
        [3],
        [4]])
>>>
```

### 3.10.3 安装

NumPy 是科学计算中最重要的 Python 包之一，它提供了高性能的数组操作和线性代数运算。NumPy 的官方网站是 <http://www.numpy.org>，提供了详细的安装和使用指南。

## 3.11 随机数

### 3.11.1 随机数

NumPy 提供了强大的随机数生成功能，可以用于模拟和数据分析。

### 3.11.2 随机数

NumPy 的 `random` 模块提供了多种随机数生成函数，其中最常用的是 `random.choice()`，用于从给定的数组中随机选择一个元素。

```
>>> import random
>>> values = [1, 2, 3, 4, 5, 6]
>>> random.choice(values)
2
>>> random.choice(values)
3
>>> random.choice(values)
1
>>> random.choice(values)
4
>>> random.choice(values)
6
>>>
```

ランダムにN個の要素を抽出する  
**random.sample()**

```
>>> random.sample(values, 2)
[6, 2]
>>> random.sample(values, 2)
[4, 3]
>>> random.sample(values, 3)
[4, 3, 1]
>>> random.sample(values, 3)
[5, 4, 1]
>>>
```

ランダムに要素を並び替える  
**random.shuffle()**

```
>>> random.shuffle(values)
>>> values
[2, 4, 6, 5, 3, 1]
>>> random.shuffle(values)
>>> values
[3, 5, 2, 1, 6, 4]
```

```
>>>
```

□□□□□□□□□□random.randint()□

```
>>> random.randint(0,10)
2
>>> random.randint(0,10)
5
>>> random.randint(0,10)
0
>>> random.randint(0,10)
7
>>> random.randint(0,10)
10
>>> random.randint(0,10)
3
>>>
```

□□□0□1□□□□□□□□□□□□□□□□  
random.random()□

```
>>> random.random()
0.9406677561675867
>>> random.random()
0.133129581343897
>>> random.random()
0.4144991136919316
>>>
```

□□□□□N□□□□□□□□□□□□□□□□  
random.getrandbits()□

```
>>> random.getrandbits(200)
335837000776573622800628485064121869519521710558559406913275
```

```
>>>
```

## 3.11.3 `random`

`random` is a module that implements a Mersenne Twister pseudorandom number generator. It provides a variety of functions for generating random numbers, including `random.seed()` for seeding the generator.

```
random.seed()           # Seed based on system time or
os.urandom()            # Seed based on integer given
random.seed(12345)      # Seed based on integer given
random.seed(b'bytedata') # Seed based on byte data
```

`random` provides several functions for generating random numbers:

- `random.random()`: Returns a random float between 0 and 1.
- `random.uniform(a, b)`: Returns a random float between `a` and `b`.
- `random.gauss(mu, sigma)`: Returns a random float from a Gaussian distribution with mean `mu` and standard deviation `sigma`.

`random` also provides a function for generating random bytes:

- `ssl.RAND_bytes(nbytes)`: Returns a random byte string of length `nbytes`.

## 3.12 `ssl`

### 3.12.1 `ssl`





```
>>> print(now)
2012-12-21 14:54:43.094063
>>> print(now + timedelta(minutes=10))
2012-12-21 15:04:43.094063
>>>
```

datetime 객체를 빼면 timedelta 객체가 반환된다.

```
>>> a = datetime(2012, 3, 1)
>>> b = datetime(2012, 2, 28)
>>> a - b
datetime.timedelta(2)
>>> (a - b).days
2
>>> c = datetime(2013, 3, 1)
>>> d = datetime(2013, 2, 28)
>>> (c - d).days
1
>>>
```

### 3.12.3 date

datetime 객체에서 날짜만 추출하면 date 객체가 반환된다. date 객체는 datetime 객체와 유사한 연산이 가능하지만, 시간과 초는 지원하지 않는다.

dateutil.relative\_delta()는 datetime과 dateutil 객체 간의 차이를 timedelta 객체로 반환한다. dateutil 객체는 dateutil.parser.parse()를 사용하여 문자열로 변환할 수 있다.

```

>>> a = datetime(2012, 9, 23)
>>> a + timedelta(months=1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'months' is an invalid keyword argument for this
function
>>>

>>> from dateutil.relativedelta import
relativedelta
>>> a + relativedelta(months=+1)
datetime.datetime(2012, 10, 23, 0, 0)
>>> a + relativedelta(months=+4)
datetime.datetime(2013, 1, 23, 0, 0)
>>>

>>> # Time between two dates

>>> b = datetime(2012, 12, 21)
>>> d = b - a
>>> d
datetime.timedelta(89)
>>> d = relativedelta(b, a)
>>> d
relativedelta(months=+2, days=+28)
>>> d.months
2
>>> d.days
28
>>>

```

## 3.13 □□□□**5**□□□

### 3.13.1 □□

Python datetime 모듈을 사용하여  
이전 요일을 찾는 방법

## 3.13.2 Python datetime

Python datetime 모듈을 사용하여  
이전 요일을 찾는 방법

```
from datetime import datetime, timedelta

weekdays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
             'Friday', 'Saturday', 'Sunday']

def get_previous_byday(dayname, start_date=None):
    if start_date is None:
        start_date = datetime.today()
    day_num = start_date.weekday()
    day_num_target = weekdays.index(dayname)
    days_ago = (7 + day_num - day_num_target) % 7
    if days_ago == 0:
        days_ago = 7
    target_date = start_date - timedelta(days=days_ago)
    return target_date
```

이전 요일을 찾는 방법

```
>>> datetime.today() # For reference

datetime.datetime(2012, 8, 28, 22, 4, 30, 263076)
>>> get_previous_byday('Monday')
datetime.datetime(2012, 8, 27, 22, 3, 57, 29045)
>>> get_previous_byday('Tuesday') # Previous week, not today

datetime.datetime(2012, 8, 21, 22, 4, 12, 629771)
```

```
>>> get_previous_byday('Friday')
datetime.datetime(2012, 8, 24, 22, 5, 9, 911393)
>>>
```

start\_date datetime

```
>>> get_previous_byday('Sunday', datetime(2012, 12, 21))
datetime.datetime(2012, 12, 16, 0, 0)
>>>
```

### 3.13.3

timedelta

python-dateutil dateutil relativedelta()

```
>>> from datetime import datetime
>>> from dateutil.relativedelta import relativedelta
>>> from dateutil.rrule import *
>>> d = datetime.now()
>>> print(d)
2012-12-23 16:31:52.718111

>>> # Next Friday
```

```
>>> print(d + relativedelta(weekday=FR))
2012-12-28 16:31:52.718111
>>>

>>> # Last Friday

>>> print(d + relativedelta(weekday=FR(-1)))
2012-12-21 16:31:52.718111
>>>
```

```
from datetime import
```

```
datetime, date, timedelta
import calendar
```

def

```
get_month_range(start_date=None):
    if
```

```
start_date is
```

None:

```

        start_date = date.today().replace(day=1)
_, days_in_month = calendar.monthrange(start_date.year,
start_date.month)
end_date = start_date + timedelta(days=days_in_month)
return

```

```
(start_date, end_date)
```

[illegible]

```
>>> a_day = timedelta(days=1)
>>> first_day, last_day = get_month_range()
>>> while
```

```
first_day < last_day:
```

```
... print
```

```
(first_day)
```

...

```
first_day += a_day
```

■ ■ ■

2012-08-01

2012-08-02

2012-08-03

2012-08-04



time 的日期时间范围，使用 `datetime.timedelta` 来指定时间间隔，小于 0 表示向后推时间。

Python 的 `range()` 函数可以生成一个数字序列，但无法生成日期时间序列。

```
def
date_range(start, stop, step):
    while
start < stop:
    yield
start
    start += step
```

使用示例：

```
>>> for
d in
date_range(datetime(2012, 9, 1), datetime(2012,10,1),
            timedelta(hours=6)):
...     print
(d)
...

2012-09-01 00:00:00
2012-09-01 06:00:00
```





```
>>>
```

### 3.15.3

`datetime.strptime()` 函数用于将字符串解析为 `datetime` 对象。其格式与 `strftime()` 类似，但使用不同的格式符。例如，`%Y` 表示年份，`%m` 表示月份，`%d` 表示日期。以下是一个示例：

以下代码将字符串 `"2012-09-23 21:37:04"` 解析为 `datetime` 对象：

```
>>> z
datetime.datetime(2012, 9, 23, 21, 37, 4, 177393)
>>> nice_z = datetime.strptime(z, '%A %B %d, %Y')
>>> nice_z
'Sunday September 23, 2012'
>>>
```

`datetime.strptime()` 函数在 Python 3.2 版本中引入。它接受两个参数：要解析的字符串和要使用的格式。格式字符串使用与 `strftime()` 相同的格式符。以下是一个示例：

“YYYY-MM-DD” 格式用于解析日期字符串。



```

>>> from datetime import datetime
>>> from pytz import timezone
>>> d = datetime(2012, 12, 21, 9, 30, 0)
>>> print(d)
2012-12-21 09:30:00
>>>

>>> # Localize the date for Chicago

>>> central = timezone('US/Central')
>>> loc_d = central.localize(d)
>>> print(loc_d)
2012-12-21 09:30:00-06:00
>>>

```

2012-12-21 09:30:00-06:00  
 2012-12-21 09:30:00-06:00

```

>>> # Convert to Bangalore time
>>> bang_d = loc_d.astimezone(timezone('Asia/Kolkata'))
>>> print

(bang_d)
2012-12-21 21:00:00+05:30
>>>

```

2012-12-21 21:00:00+05:30  
 2012-12-21 21:00:00+05:30  
 2012-12-21 21:00:00+05:30  
 2012-12-21 21:00:00+05:30

```

>>> d = datetime(2013, 3, 10, 1, 45)
>>> loc_d = central.localize(d)
>>> print

```

```
(loc_d)
2013-03-10 01:45:00-06:00
>>> later = loc_d + timedelta(minutes=30)
>>> print

(later)
2013-03-10 02:15:00-06:00      # WRONG! WRONG!
>>>
```

1
 `central.normalize()`

```
>>> from datetime import timedelta
>>> later = central.normalize(loc_d + timedelta(minutes=30))
>>> print(later)
2013-03-10 03:15:00-05:00
>>>
```

### 3.16.3

UTC
 UTC

```
>>> print(loc_d)
2013-03-10 01:45:00-06:00
>>> utc_d = loc_d.astimezone(pytz.utc)
>>> print(utc_d)
2013-03-10 07:45:00+00:00
>>>
```

**0000UTC**

```
>>> later_utc = utc_d + timedelta(minutes=30)
>>> print
(later_utc.astimezone(central))
2013-03-10 03:15:00-05:00
>>>
```

```
pytz.country_timezones["Asia/Kolkata"]
```

```
>>> pytz.country_timezones['IN']
['Asia/Kolkata']
>>>
```

```

PEP 431
pytz
UTC

```

[1] 06

## 4 迭代器

Python 中，迭代器就是实现了 `__iter__()` 和 `__next__()` 方法的对象。Python 中，`iter()` 函数返回一个迭代器对象，而 `next()` 函数返回下一个元素。Python 中，`iter()` 函数返回一个迭代器对象，而 `next()` 函数返回下一个元素。Python 中，`iter()` 函数返回一个迭代器对象，而 `next()` 函数返回下一个元素。

### 4.1 迭代器

#### 4.1.1 列表

列表是一个可迭代的对象，我们可以使用 `for` 循环来遍历列表中的元素。

#### 4.1.2 生成器

生成器是一种特殊的迭代器，它可以在需要的时候生成下一个元素，而不是在需要的时候生成下一个元素。生成器使用 `yield` 语句来生成元素，而不是使用 `return` 语句。生成器使用 `yield` 语句来生成元素，而不是使用 `return` 语句。

```
with  
open('/etc/passwd') as  
f:  
    try
```

```

:
    while
True:
    line = next(f)
    print
(line, end='')
    except StopIteration
:
    pass

```

StopIteration
 next()
 None

```

with
open('/etc/passwd') as
f:
    while
True:
    line = next(f, None)
    if
line is
None:
    break
    print

```



```
(line, end='')
```

## 4.1.3 for

for loops are used to iterate over a sequence of elements. The sequence can be a list, a tuple, a string, or any other iterable object. The loop variable is assigned the value of the element being iterated over.

for loops are used to iterate over a sequence of elements.

```
>>> items = [1, 2, 3]
>>> # Get the iterator

>>> it = iter(items)      # Invokes items.__iter__()

>>> # Run the iterator

>>> next(it)              # Invokes it.__next__()

1
>>> next(it)
2
>>> next(it)
3
>>> next(it)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
>>>
```

이제 이 코드를 실행하면 다음과 같은 결과가 출력됩니다.

## 4.2 클래스

### 4.2.1 클래스

이제 이 코드를 실행하면 다음과 같은 결과가 출력됩니다.

### 4.2.3 클래스

이제 이 코드를 실행하면 다음과 같은 결과가 출력됩니다.

```
class Node:
    def __init__(self, value):
        self._value = value
        self._children = []

    def __repr__(self):
        return 'Node({!r})'.format(self._value)

    def add_child(self, node):
        self._children.append(node)

    def __iter__(self):
        return iter(self._children)

# Example

if __name__ == '__main__':
```

```

root = Node(0)
child1 = Node(1)
child2 = Node(2)
root.add_child(child1)
root.add_child(child2)
for ch in root:
    print(ch)
# Outputs Node(1), Node(2)

```

Python의 `__iter__()` 메서드는 객체가 반복 가능한 객체임을 나타내며, `__children__` 속성을 반환합니다.

## 4.2.4 Python의 Iterable

Python에서 `__iter__()` 메서드를 가진 객체는 `Iterable` 인터페이스를 구현한 것으로 간주됩니다. 이 인터페이스는 `__next__()` 메서드를 포함하며, 이 메서드는 다음 요소를 반환하거나 `StopIteration` 예외를 발생시킵니다.

Python에서 `iter()` 함수는 객체를 `Iterator` 객체로 변환합니다. `iter(s)`는 `s.__iter__()`를 호출하여 `Iterator` 객체를 반환합니다. `len(s)`는 `s.__len__()`를 호출하여 객체의 길이를 반환합니다.

## 4.3 Python의 Sequence

### 4.3.1 Python의 Sequence

range() reversed ()

## 4.3.2

```
def
frange(start, stop, increment):
    x = start
    while
x < stop:
    yield
x
    x += increment
```

for sum() list()

```
>>> for
n in
frange(0, 4, 0.5):
...     print
(n)
...
```

```

0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
>>> list(frange(0, 1, 0.125))
[0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875]
>>>

```

### 4.3.3 yield

yield is a keyword that is used to make a function a generator. It is used to return a sequence of values from a function. It is used to return a sequence of values from a function. It is used to return a sequence of values from a function.

```

>>> def
countdown(n):
...     print
('Starting to count from', n)
...     while
n > 0:
...         yield
n
...
n -= 1
...     print
('Done!')
...

```

```

    "next"
    for

```

## 4.4 探索搜索

### 4.4.1 广度

广度搜索从根节点开始，逐层遍历所有子节点，直到找到目标节点。这种搜索方式适用于寻找最短路径的问题。

### 4.4.2 深度

深度搜索从根节点开始，沿着一条路径一直遍历下去，直到达到叶子节点。如果该叶子节点不是目标节点，则回溯到父节点，继续遍历下一个子节点。这种搜索方式适用于寻找所有可能的解的问题。

4.2 实现一个 Node 类，用于表示树中的节点。

```
class Node:
    def __init__(self, value):
        self._value = value
        self._children = []

    def __repr__(self):
        return 'Node({!r})'.format(self._value)

    def add_child(self, node):
        self._children.append(node)

    def __iter__(self):
        return iter(self._children)

    def depth_first(self):
        yield self
        for c in self:
            yield from c.depth_first()

# Example
```

```

if __name__ == '__main__':
    root = Node(0)
    child1 = Node(1)
    child2 = Node(2)
    root.add_child(child1)
    root.add_child(child2)
    child1.add_child(Node(3))
    child1.add_child(Node(4))
    child2.add_child(Node(5))

    for ch in root.depth_first():
        print(ch)
    # Outputs Node(0), Node(1), Node(3), Node(4), Node(2),
    Node(5)

```

depth\_first() returns an iterator  
 that yields from depth\_first()

### 4.4.3 Iterator

Python defines `__iter__()` and `__next__()` methods to implement an iterator.  
 The `__next__()` method returns the next element or raises `StopIteration` if  
 there are no more elements. The `depth_first()` method returns an iterator  
 that yields from `depth_first()`.

```

class Node:
    def __init__(self, value):
        self._value = value

```



```

        self._children = []

def __repr__(self):
    return 'Node({!r})'.format(self._value)

def add_child(self, other_node):
    self._children.append(other_node)

def __iter__(self):
    return iter(self._children)

def depth_first(self):
    return DepthFirstIterator(self)

class DepthFirstIterator(object):
    '''
Depth-first traversal

    '''

def __init__(self, start_node):
    self._node = start_node
    self._children_iter = None
    self._child_iter = None

def __iter__(self):
    return self

def __next__(self):
    # Return myself if just started; create an iterator for children

    if self._children_iter is None:
        self._children_iter = iter(self._node)
        return self._node

    # If processing a child, return its next item

    elif self._child_iter:

```

```
try:
    nextchild = next(self._child_iter)
    return nextchild
except StopIteration:
    self._child_iter = None
    return next(self)

# Advance to the next child and start its iteration

else:
    self._child_iter =
next(self._children_iter).depth_first()
    return next(self)
```

DepthFirstIterator

# 4.5

## 4.5.1

## 4.5.2

reversed()

```

>>> a = [1, 2, 3, 4]
>>> for
x in
reversed(a):
...     print
(x)
...

4
3
2
1

```

reversed()

```

# Print a file backwards
f = open('somefile')
for
line in
reversed(list(f)):
    print
(line, end='')

```

reversed()

## 4.5.3

Countdown class with `__reversed__()` method

```
class Countdown:
    def __init__(self, start):
        self.start = start
        # Forward iterator

    def __iter__(self):
        n = self.start
        while n > 0:
            yield n
            n -= 1

    # Reverse iterator

    def __reversed__(self):
        n = 1
        while n <= self.start:
            yield n
            n += 1
```

Countdown class with `__reversed__()` method

## 4.6

### 4.6.1

이 코드는 Python 3.5 이상에서 동작하며, Python 2에서는 Python 2.7.9 이상에서 동작합니다. Python 2.7.9 이하에서는 collections.deque를 사용하지 않고, list를 사용하여 구현할 수 있습니다.

## 4.6.2 linehistory

이 코드는 Python 3.5 이상에서 동작하며, Python 2에서는 Python 2.7.9 이상에서 동작합니다. Python 2.7.9 이하에서는 collections.deque를 사용하지 않고, list를 사용하여 구현할 수 있습니다.

```
from collections import deque

class linehistory:
    def __init__(self, lines, histlen=3):
        self.lines = lines
        self.history = deque(maxlen=histlen)

    def __iter__(self):
        for lineno, line in enumerate(self.lines, 1):
            self.history.append((lineno, line))
            yield line

    def clear(self):
        self.history.clear()
```

이 코드는 Python 3.5 이상에서 동작하며, Python 2에서는 Python 2.7.9 이상에서 동작합니다. Python 2.7.9 이하에서는 collections.deque를 사용하지 않고, list를 사용하여 구현할 수 있습니다.

```
with
open('somefile.txt') as
f:
    lines = linehistory(f)
```

```

        for
line in
lines:
    if
'python' in
line:
        for
lineno, hline in
lines.history:
            print
('{}:{}'.format(lineno, hline), end='')

```

## 4.6.3 `__iter__`

`__iter__` is a method that returns an iterator object. This object has a `__next__` method that returns the next item in the sequence. The `__iter__` method is called when the object is passed to the `iter()` function.

The `__iter__` method is called when the object is passed to the `iter()` function.

```

>>> f = open('somefile.txt')
>>> lines = linehistory(f)
>>> next(lines)

```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'linehistory' object is not an iterator
```

```
>>> # Call iter() first, then start iterating
```

```
>>> it = iter(lines)
>>> next(it)
'hello world\n'
>>> next(it)
'this is a test\n'
>>>
```

## 4.7 `iter`

### 4.7.1 `iter`

`iter` is a built-in function that takes an object and returns an iterator object. The object must implement the `__iter__` method, which returns the iterator object. The iterator object implements the `__next__` method, which returns the next value from the iterator.

### 4.7.2 `islice`

`itertools.islice()` is a function that returns an iterator that yields a slice of the input iterable. The slice is defined by the start, stop, and step arguments. The start argument is the index of the first element to yield. The stop argument is the index of the last element to yield. The step argument is the distance between elements to yield.

```
>>> def
count(n):
...     while
True:
...         yield
```

```
n
...         n += 1
...
>>> c = count(0)
>>> c[10:20]
Traceback (most recent call last):
  File "<stdin>", line 1, in
```

```
<module>
TypeError
```

```
: 'generator' object is not
subscriptable
```

```
>>> # Now using islice()
```

```
>>> import itertools
```

```
>>> for
```

```
x in
```

```
itertools.islice(c, 10, 20):
...     print
```

```
(x)
```

```
...
```

```
10
```

```
11
```

```
12
```

```
13
```

```
14
```

```
15
```

```
16
```

```
17
```

```
18
```

```
19
```

```
>>>
```



### 4.7.3 `islice`

`islice` 是 `slice` 的惰性版本。它返回一个惰性切片对象，该对象在迭代时才会生成切片中的元素。它接受与 `slice` 相同的参数，并返回一个惰性切片对象。它返回的切片对象在迭代时才会生成切片中的元素。

`islice` 的用法与 `slice` 类似。它接受起始、结束和步长的参数，并返回一个惰性切片对象。它返回的切片对象在迭代时才会生成切片中的元素。

## 4.8 `itertools`

### 4.8.1 `dropwhile`

`dropwhile` 是一个函数，它接受一个可迭代对象和一个谓词函数。它返回一个惰性切片对象，该对象在迭代时才会生成切片中的元素。它返回的切片对象在迭代时才会生成切片中的元素。

### 4.8.2 `dropwhile`

`dropwhile` 是一个函数，它接受一个可迭代对象和一个谓词函数。它返回一个惰性切片对象，该对象在迭代时才会生成切片中的元素。它返回的切片对象在迭代时才会生成切片中的元素。

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  
XXXXXX

```
>>> with
open('/etc/passwd') as
f:
...     for
line in
f:
...     print
(line, end='')
...

##
# User Database
#
# Note that this file is consulted directly only when the
system is running
# in single-user mode. At other times, this information is
provided by
# Open Directory.
...

##
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
root:*:0:0:System Administrator:/var/root:/bin/sh
...

>>>
```

XXXXXXXXXXXXXXXXXXXXXXXXXXXX

```
>>> from itertools import dropwhile
>>> with open('/etc/passwd') as f:
...     for line in dropwhile(lambda line:
line.startswith('#'), f):
...         print(line, end='')
...
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
root:*:0:0:System Administrator:/var/root:/bin/sh
...
>>>
```

itertools.islice()

```
>>> from itertools import islice
>>> items = ['a', 'b', 'c', 1, 4, 10, 15]
>>> for x in islice(items, 3, None):
...     print(x)
...
1
4
10
15
>>>
```

itertools.islice()
 None
 3
 3
 [3:]
 [:3]

### 4.8.3

dropwhile()
 islice()

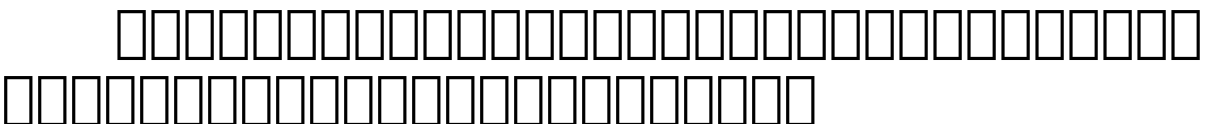
```
with
open('/etc/passwd') as
f:
    # Skip over initial comments

    while
True:
        line = next(f, '')
        if not
line.startswith('#'):
            break

# Process remaining lines

while
line:
    # Replace with useful processing

    print
(line, end='')
    line = next(f, None)
```



```
with
open( '/etc/passwd' ) as
```

```
f:
    lines = (line for
line in
f if not
line.startswith('#'))
    for
line in
lines:
    print
(line, end='')
```

The diagram consists of three horizontal rows of boxes. The top row contains 20 boxes, the middle row contains 25 boxes, and the bottom row contains 20 boxes. The boxes are arranged in a staggered pattern: the middle row starts at the left edge, while the top and bottom rows are centered relative to the middle row's span.

## 4.9

### 4.9.1 〇〇

[illegible]

## 4.9.2 itertools

itertools 3  
itertools.permutations() —  
permutations() —  
permutations() —  
permutations() —

```
>>> items = ['a', 'b', 'c']
>>> from itertools import
permutations
>>> for
p in
permutations(items):
...     print
(p)
...
('a', 'b', 'c')
('a', 'c', 'b')
('b', 'a', 'c')
('b', 'c', 'a')
('c', 'a', 'b')
('c', 'b', 'a')
>>>
```

permutations() —  
permutations() —

```
>>> for
p in
```

```
permutations(items, 2):
...     print
    (p)
...

('a', 'b')
('a', 'c')
('b', 'a')
('b', 'c')
('c', 'a')
('c', 'b')
>>>
```

```
from itertools import combinations
```

```
>>> from itertools import combinations
>>> for c in combinations(items, 3):
...     print(c)
...
('a', 'b', 'c')
>>> for c in combinations(items, 2):
...     print(c)
...
('a', 'b')
('a', 'c')
('b', 'c')
>>> for c in combinations(items, 1):
...     print(c)
...
('a',)
('b',)
('c',)
>>>
```

`combinations()` returns combinations of the items in the iterable, without replacement. For example, `combinations('a', 'b')` returns `('a', 'b')` but not `('b', 'a')`.

`combinations_with_replacement()` returns combinations of the items in the iterable, with replacement. For example, `combinations_with_replacement('a', 3)` returns `('a', 'a', 'a')`.

```
>>> for
c in
combinations_with_replacement(items, 3):
...     print
(c)
...

('a', 'a', 'a')
('a', 'a', 'b')
('a', 'a', 'c')
('a', 'b', 'b')
('a', 'b', 'c')
('a', 'c', 'c')
('b', 'b', 'b')
('b', 'b', 'c')
('b', 'c', 'c')
('c', 'c', 'c')
>>>
```

## 4.9.3



itertools itertools  
 itertools itertools  
 itertools itertools  
 itertools

## 4.10 -

### 4.10.1

### 4.10.2

enumerate()

```
>>> my_list = ['a', 'b', 'c']
>>> for
idx, val in
enumerate(my_list):
...     print
(idx, val)
...

0 a
1 b
2 c
```

start

```
>>> my_list = ['a', 'b', 'c']
>>> for
    idx, val in
        enumerate(my_list, 1):
            print
                (idx, val)
            ...

1 a
2 b
3 c
```

```
def
parse_data(filename):
    with
open(filename, 'rt') as
f:
    for
lineno, line in
enumerate(f, 1):
    fields = line.split()
    try
```



```
word in
words:
    word_summary[word].append(idx)
```

```

word_summary
defaultdict
2

```

### 4.10.3 ☐☐

```

    enumerate()

```

```
lineno = 1
for
line in
f:
    # Process line
...
lineno += 1
```

```
enum enumerate()
```

for	
-----	--

```
lineno, line in
enumerate(f):
    # Process line

...
```

enumerate() returns an enumerate object  
which is an iterable object and can be looped over  
next() returns the next element

enumerate() returns an enumerate object  
which is an iterable object and can be looped over

```
data = [ (1, 2), (3, 4), (5, 6), (7, 8) ]

# Correct!

for
n, (x, y) in
enumerate(data):
    ...
# Error!

for
n, x, y in
enumerate(data):
```

...

## 4.11 字典的遍历

### 4.11.1 遍历字典

遍历字典的常用方法有：  
1. 遍历字典的键  
2. 遍历字典的值  
3. 遍历字典的键值对

### 4.11.2 遍历字典的键值对

遍历字典的键值对，可以使用 `zip()` 函数。

```
>>> xpts = [1, 5, 4, 2, 10, 7]
>>> ypts = [101, 78, 37, 15, 62, 99]
>>> for
x, y in
zip(xpts, ypts):
...     print
(x,y)
...

1 101
5 78
4 37
2 15
10 62
7 99
>>>
```

`zip(a, b)` returns an iterator of tuples where each tuple contains one element from `a` and one element from `b`. The iterator stops when the shortest input iterable is exhausted. This is useful for combining elements from multiple iterables into pairs or groups.

```
>>> a = [1, 2, 3]
>>> b = ['w', 'x', 'y', 'z']
>>> for
    i in
zip(a,b):
...     print
    (i)
...

(1, 'w')
(2, 'x')
(3, 'y')
>>>
```

`itertools.zip_longest()` returns an iterator of tuples where each tuple contains one element from `a` and one element from `b`. The iterator stops when the longest input iterable is exhausted. This is useful for combining elements from multiple iterables into pairs or groups, even if the iterables have different lengths.

```
>>> from itertools import zip_longest
>>> for i in zip_longest(a,b):
...     print(i)
...
(1, 'w')
(2, 'x')
(3, 'y')
(None, 'z')
>>> for i in zip_longest(a, b, fillvalue=0):
...     print(i)
...
(1, 'w', 0)
(2, 'x', 0)
(3, 'y', 0)
(None, 'z', 0)
```

```
(1, 'w')
(2, 'x')
(3, 'y')
(0, 'z')
>>>
```

## 4.11.3 zip

`zip()` returns an iterator of tuples, where the i-th tuple contains the i-th element from each of the argument sequences or iterables. The number of tuples returned is determined by the length of the shortest argument iterable.

```
headers = ['name', 'shares', 'price']
values = ['ACME', 100, 490.1]
```

`zip(headers, values)` returns an iterator of tuples, where the i-th tuple contains the i-th element from each of the argument sequences or iterables.

```
s = dict(zip(headers, values))
```

`dict(zip(headers, values))` returns a dictionary with the same elements as the dictionary created by `s = dict(zip(headers, values))`.

```
for
name, val in
zip(headers, values):
    print
(name, '=', val)
```



zip() 2  
zip() 2

```
>>> a = [1, 2, 3]
>>> b = [10, 11, 12]
>>> c = ['x', 'y', 'z']
>>> for
```

```
i in
```

```
zip(a, b, c):
...     print
```

```
(i)
...
```

```
(1, 10, 'x')
(2, 11, 'y')
(3, 12, 'z')
>>>
```

zip() list()

```
>>> zip(a, b)
<zip object at 0x1007001b8>
>>> list(zip(a, b))
[(1, 10), (2, 11), (3, 12)]
>>>
```

## 4.12

## 4.12.1

## 4.12.2

`itertools.chain()`

```
>>> from itertools import chain
>>> a = [1, 2, 3, 4]
>>> b = ['x', 'y', 'z']
>>> for x in chain(a, b):
...     print(x)
...
1
2
3
4
x
y
z
>>>
```

`chain()`

```
# Various working sets of items
active_items = set()
```

```
inactive_items = set()

# Iterate over all items
for
    item in
        chain(active_items, inactive_items):
            # Process item
            ...
```

```
chain()
```

```

for

item in

active_items:
    # Process item

...

for

item in

inactive_items:
    # Process item

...

```

### 4.12.3 □□

`itertools.chain()`은 여러 iterable 객체를 하나로 합쳐주는 함수이다. `chain()`은 여러 iterable 객체를 하나로 합쳐주는 함수이다. `chain()`은 여러 iterable 객체를 하나로 합쳐주는 함수이다.

```
# Inefficient
for
x in
a + b:
...

# Better
for
x in
chain(a, b):
...

```

`a + b`는 `a`와 `b`를 합친 새로운 iterable 객체를 생성한다. `chain()`은 `a`와 `b`를 합친 새로운 iterable 객체를 생성한다. `chain()`은 `a`와 `b`를 합친 새로운 iterable 객체를 생성한다.

## 4.13 itertools.chain()

## 4.13.1 目录

目录是文件系统的重要组成部分，它记录了文件系统的结构。UNIX 目录结构是树形的，根目录是 /，下面是一级目录，一级目录下是二级目录，以此类推。目录结构如下：

## 4.13.2 目录结构

目录结构是树形的，根目录是 /，下面是一级目录，一级目录下是二级目录，以此类推。目录结构如下：

```
foo/  
  access-log-012007.gz  
  access-log-022007.gz  
  access-log-032007.gz  
  ...  
  access-log-012008  
bar/  
  access-log-092007.bz2  
  ...  
  access-log-022008
```

目录结构是树形的，根目录是 /，下面是一级目录，一级目录下是二级目录，以此类推。目录结构如下：

```
124.115.6.12 - - [10/Jul/2012:00:18:50 -0500] "GET /robots.txt  
..." 200 71  
210.212.209.67 - - [10/Jul/2012:00:18:51 -0500] "GET /ply/  
..." 200 11875  
210.212.209.67 - - [10/Jul/2012:00:18:51 -0500] "GET  
/favicon.ico ..." 404 369  
61.135.216.105 - - [10/Jul/2012:00:20:04 -0500] "GET  
/blog/atom.xml ..." 304 -
```

...

██  
██

```
import os
import fnmatch
import gzip
import bz2
import re

def gen_find(filepat, top):
    '''
        Find all filenames in a directory tree that match a shell
        wildcard pattern

        ...

        for path, dirlist, filelist in os.walk(top):
            for name in fnmatch.filter(filelist, filepat):
                yield os.path.join(path,name)

def gen_opener(filenames):
    '''
        Open a sequence of filenames one at a time producing a
        file object.

        The file is closed immediately when proceeding to the next
        iteration.

        ...
```

```

for filename in filenames:
    if filename.endswith('.gz'):
        f = gzip.open(filename, 'rt')
    elif filename.endswith('.bz2'):
        f = bz2.open(filename, 'rt')
    else:
        f = open(filename, 'rt')
    yield f
    f.close()

def gen_concatenate(iterators):
    '''
    Chain a sequence of iterators together into a single
    sequence.

    ...

    for it in iterators:
        yield from it

def gen_grep(pattern, lines):
    '''
    Look for a regex pattern in a sequence of lines

    ...

    pat = re.compile(pattern)
    for line in lines:
        if pat.search(line):
            yield line

```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
 □□□□□□□□□python□□□□□□□□□□□□□□□□□□□□□□





```
    for line in file:
        yield line
```

```
def gen_concatenate(files):
    yield from (
        gen_opener(file)
        for file in files
    )
    sum(chain(*files))
```

```
def gen_opener(file):
    with open(file) as f:
        yield f
```

```
def gen_concatenate(files):
    yield from (
        gen_opener(file)
        for file in files
    )
```

```
def gen_concatenate(files):
    yield from (
        gen_opener(file)
        for file in files
    )
    lines = itertools.chain(*files)
    gen_opener(chain(*files))
```

```
def gen_concatenate():
    yield from it
gen_concatenate()
4.14
```

David Beazley “<http://www.dabeaz.com/generators>”

## 4.14

### 4.14.1

### 4.14.2

```
yield from
```

```
from collections import Iterable
```

```

def flatten(items, ignore_types=(str, bytes)):
    for x in items:
        if isinstance(x, Iterable) and not isinstance(x,
ignore_types):
            yield from flatten(x)
        else:
            yield x

items = [1, 2, [3, 4, [5, 6], 7], 8]

# Produces 1 2 3 4 5 6 7 8

for x in flatten(items):
    print(x)

```

```

    if isinstance(x, Iterable) and not
yield from flatten(x)

```

```

    ignore_types and not
isinstance(x, ignore_types) and not

```

```

>>> items = ['Dave', 'Paula', ['Thomas', 'Lewis']]
>>> for
x in
flatten(items):
...
    print

```

```
(x)
...

Dave
Paula
Thomas
Lewis
>>>
```

## 4.14.3

yield from  
for

```
def
flatten(items, ignore_types=(str, bytes)):
    for
x in
items:
    if
isinstance(x, Iterable) and not
isinstance(x, ignore_types):
        for
i in
flatten(x):
            yield
i
```

```

        else
:
        yield
x

```

yield from

ignore\_types

yield from
 coroutine
 12.12

## 4.15

### 4.15.1

yield from

### 4.15.2

heapq.merge() returns an iterator that yields the elements of the sorted lists in order.

```
>>> import heapq
>>> a = [1, 4, 7, 10]
>>> b = [2, 5, 6, 11]
>>> for c in heapq.merge(a, b):
...     print(c)
...
1
2
4
5
6
7
10
11
```

### 4.15.3

heapq.merge() can be used to merge multiple sorted files into a single sorted file.

```
import heapq

with open('sorted_file_1', 'rt') as file1, \
     open('sorted_file_2', 'rt') as file2, \
     open('merged_file', 'wt') as outf:

    for line in heapq.merge(file1, file2):
        outf.write(line)
```

heapq.merge() 合并多个有序序列  
 heapq.merge() 返回一个生成器，该生成器返回一个有序序列。  
 heapq.merge() 的复杂度是 O(n log k)，其中 n 是所有序列的总长度，k 是序列的数量。  
 heapq.merge() 可以用于合并多个有序列表，例如在归并排序中。  
 heapq.merge() 也可以用于合并多个有序流，例如在数据库查询中。  
 heapq.merge() 是一个非常有用的函数，特别是在处理大规模数据时。

## 4.16 while

### 4.16.1

while 语句用于循环执行一段代码，直到条件为假为止。  
 while 语句的语法如下：

### 4.16.2

I/O 操作通常使用 while 循环。

```
CHUNKSIZE = 8192

def
reader(s):
    while
True:
        data = s.recv(CHUNKSIZE)
        if
data == b'':
```

```
break
```

```
process_data(data)
```

```
def iter():
```

```
def
```

```
reader(s):
```

```
for
```

```
chunk in
```

```
iter(lambda
```

```
: s.recv(CHUNKSIZE), b''):  
    process_data(data)
```

```
def iter():  
    while True:  
        chunk = reader(s)  
        yield chunk
```

```
>>> import sys  
>>> f = open('/etc/passwd')  
>>> for chunk in iter(lambda: f.read(10), ''):  
...     n = sys.stdout.write(chunk)  
...  
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false  
root:*:0:0:System Administrator:/var/root:/bin/sh  
daemon:*:1:1:System Services:/var/root:/usr/bin/false  
_uucp:*:4:4:Unix to Unix Copy  
Protocol:/var/spool/uucp:/usr/sbin/uucico  
...  
>>>
```



## 4.16.3 迭代器

Python 的 `iter()` 函数返回一个迭代器对象，该对象实现了 `__iter__()` 方法，返回一个迭代器对象。该对象实现了 `__next__()` 方法，返回下一个元素。当没有更多元素时，就会引发 `StopIteration` 异常。

Python 的 `iter()` 函数可以用于任何实现了 `__iter__()` 方法的对象。例如，字符串、列表、字典、文件对象、`socket` 对象等。对于 `socket` 对象，`read()` 和 `recv()` 方法返回的数据可以被视为一个迭代器。使用 `iter()` 函数可以方便地遍历数据。例如，使用 `lambda` 表达式可以创建一个返回 `recv()` 或 `read()` 方法的函数。

---

[1] 本文档中的代码是在 Python 3.6.0 版本下测试通过的。在 Python 3.5.0 版本下，`socket` 对象的 `recv()` 方法返回的数据不能被视为一个迭代器。

# 5 I/O

Python has a rich set of I/O functions and modules that allow you to read and write files and streams. This section covers the basics of file I/O, including opening files, reading and writing data, and closing files.

## 5.1 File Objects

### 5.1.1 Opening Files

Files are opened using the `open()` function, which returns a file object. The file object can then be used to read or write data. The `open()` function takes two arguments: the filename and the mode. The mode can be 'r' for read, 'w' for write, 'a' for append, 'r+' for read and write, and 'w+' for write and read.

### 5.1.2 Reading Files

The `open()` function can be used to open a file in read mode ('r'). The file object can then be used to read data from the file. The `read()` method reads the entire file as a single string. The `readlines()` method reads the file line by line, returning a list of strings.

```
# Read the entire file as a single string
```

```
with
```

```
open('somefile.txt', 'rt') as
```

```
f:
```

```
    data = f.read()
```

```
# Iterate over the lines of the file
```

**with**

```
open('somefile.txt', 'rt') as
```

 $f:$ 

for

line in

 $f:$ 

```
# process line
```

■ ■ ■

```

open()wt

```

```
# Write chunks of text data
```

**with**

```
open('somefile.txt', 'wt') as
```

 $f:$ 

```
f.write(text1)
```

```
f.write(text2)
```

■ ■ ■

```
# Redirected print statement
```

```
with
open('somefile.txt', 'wt') as
f:
    print
(line1, file=f)
    print
(line2, file=f)
    ...
```

```

    open() at

```

```

import sys
sys.setdefaultencoding('utf-8')
f = open('test.txt', 'w')

```

```
with
open('somefile.txt', 'rt', encoding='latin-1') as
f:
    ...
```



```

f = open('data.txt', 'w')
f.write('hello\n')

```

```
# Read with disabled newline translation

with
open('somefile.txt', 'rt', newline='') as
f:
    ...
```

```

    _____UNIX____
Windows_____hello
world!\r\n_____

```

```
>>> # Newline translation enabled (the default)

>>> f = open('hello.txt', 'rt')
>>> f.read()
'hello world!\n'
>>> # Newline translation disabled

>>> g = open('hello.txt', 'rt', newline='')
>>> g.read()
'hello world!\r\n'
>>>
```



```
>>> g.read()
'Spicy Jalapeo!'
>>>
```

When you call `open()` with an `encoding` and `errors` argument, Python will use those arguments to handle encoding hacks that you can find in the Python documentation. The default encoding is `utf-8`.

## 5.2 File Objects

### 5.2.1 Opening Files

The `print()` function prints to the standard output.

### 5.2.2 File Objects

The `print()` function prints to the standard output. The `file` argument is optional and defaults to `sys.stdout`.

```
with
open('somefile.txt', 'rt') as
f:
    print
('Hello World!', file=f)
```



## 5.2.3 练习

编写一个程序，计算并打印出1到100之间所有偶数的平方和。

## 5.3 字符串

### 5.3.1 打印

使用print()函数打印以下字符串：

### 5.3.2 格式化

使用print()函数，通过sep和end参数，打印以下字符串：

```
>>> print
('ACME', 50, 91.5)
ACME 50 91.5
>>> print
('ACME', 50, 91.5, sep=',')
ACME,50,91.5
>>> print
```

```
('ACME', 50, 91.5, sep=',', end='!!\n')
ACME,50,91.5!!
>>>
```

end

```
>>> for
i in
range(5):
...     print
(i)
...

0
1
2
3
4
>>> for
i in
range(5):
...     print
(i, end=' ')
...

0 1 2 3 4 >>>
```

### 5.3.3

print()  
sep  
str.join()

```
>>> print  
  
(' ','.join('ACME','50','91.5'))  
ACME,50,91.5  
>>>
```

str.join()  
ACME,50,91.5

```
>>> row = ('ACME', 50, 91.5)  
>>> print  
  
(' ','.join(row))  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: sequence item 1: expected str instance, int found  
>>> print  
  
(' ','.join(str(x) for  
x in  
row))  
ACME,50,91.5  
>>>
```

print()

```
>>> print
(*row, sep=',')
ACME,50,91.5
>>>
```

## 5.4 文件操作

### 5.4.1 文件

文件操作是Python中非常基础且重要的部分，本章将详细介绍文件操作的相关知识。

### 5.4.2 文件操作

Python提供了丰富的文件操作函数，其中最常用的是`open()`函数。该函数用于打开文件，并返回一个文件对象。文件对象提供了多种方法来读取和写入文件内容。

```
# Read the entire file as a single byte string

with
open('somefile.bin', 'rb') as
f:
    data = f.read()
# Write binary data to a file

with
open('somefile.bin', 'wb') as
```

```
f:
    f.write(b'Hello World')
```

byte  
string  
bytearray

### 5.4.3

```
>>> # Text string
```

```
>>> t = 'Hello World'
```

```
>>> t[0]
```

```
'H'
```

```
>>> for
```

```
c in
```

```
t:
```

```
...     print
```

```
(c)
```

```
...
```

|| Hello

```
>>> # Byte string
```

```
>>> b = b'Hello World'
```

```
>>> b[0]
```

72

```
>>> for
```

**c in**

b:

```
... print
```

(c)

□ □ □

72

101

108

108

111

...

>>>

**with**

```

open('somefile.bin', 'rb') as
f:
    data = f.read(16)
    text = data.decode('utf-8')

with
open('somefile.bin', 'wb') as
f:
    text = 'Hello World'
    f.write(text.encode('utf-8'))

```

Python I/O 与 C 语言  
 Python 的 `byte` 类型

```

import array

nums = array.array('i', [1, 2, 3, 4])
with
open('data.bin', 'wb') as
f:
    f.write(nums)

```

Python 的 `buffer` 接口  
 Python 的 `buffer` 接口

readinto() 方法

```
>>> import array

>>> a = array.array('i', [0, 0, 0, 0, 0, 0, 0, 0])
>>> with
open('data.bin', 'rb') as
f:
...
f.readinto(a)
...

16
>>> a
array('i', [1, 2, 3, 4, 0, 0, 0, 0])
>>>
```

word 5.9 mutable buffer

## 5.5

### 5.5.1



open()은 파일을 열고, write()은 파일을 쓰는 함수이다.  
w: write, x: create, a: append

## 5.5.2 파일 쓰기

open()은 파일을 열고, write()은 파일을 쓰는 함수이다.  
w: write, x: create, a: append

```
>>> with
open('somefile', 'wt') as
f:
...
f.write('Hello\n
')
...

>>> with
open('somefile', 'xt') as
f:
...
f.write('Hello\n
')
...

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileExistsError: [Errno 17] File exists: 'somefile'
>>>
```

### 5.5.3 ☐ ☐

```
>>> import os

>>> if not
os.path.exists('somefile'):
...     with
open('somefile', 'wt') as
f:
...
f.write('Hello\n
')
...     else
:
...         print
('File already exists!')
...

File already exists!
>>>
```



```

>>>
>>> # Wrap a file interface around an existing string

>>> s = io.StringIO('Hello\n
World\n
')
>>> s.read(4)
'Hell'
>>> s.read()
'o\nWorld\n'
>>>

```

`io.StringIO`은 문자열을 읽고 쓰는 객체로, `io.BytesIO`는 바이트 데이터를 처리하는 객체이다.

```

>>> s = io.BytesIO()
>>> s.write(b'binary data')
>>> s.getvalue()
b'binary data'
>>>

```

### 5.6.3 파일 객체

`io.StringIO`와 `io.BytesIO`는 메모리 상에 데이터를 저장하는 객체이지만, 실제 파일과 유사한 인터페이스를 제공하는 `io.BufferedReader`와 `io.BufferedWriter`도 있다. 이들은 `StringIO`와 `BytesIO`를 상속받으며, `read()`, `write()`, `seek()` 등의 메서드를 제공한다.

StringIOBytesIO  
 StringIOBytesIO  
 StringIOBytesIO  
 StringIOBytesIO

## 5.7 StringIOBytesIO

### 5.7.1 StringIOBytesIO

StringIOBytesIOgzipbz2

### 5.7.2 StringIOBytesIO

gzipbz2  
 StringIOBytesIOopen()  
 StringIOBytesIO

```
# gzip compression

import gzip

with
gzip.open('somefile.gz', 'rt') as
f:
    text = f.read()
# bz2 compression
```

```
import bz2

with
bz2.open('somefile.bz2', 'rt') as
f:
    text = f.read()
```

□□□□□□□□□□□□□□□□□□□□

```
# gzip compression
import gzip

with

gzip.open('somefile.gz', 'wt') as f:
    f.write(text)

# bz2 compression
import bz2

with

bz2.open('somefile.bz2', 'wt') as f:
    f.write(text)
```

□□□□□□□□□□□□□□I/O□□□□□□□□□□□□□□□□  
Unicode□□/□□□□□□□□□□□□□□□□□□□□rb□wb□  
□□

## 5.7.3 圧縮

gzip.open() bz2.open() は、`open()` のオプションとして `encoding`、`errors`、`newline` を指定できる。

`compresslevel` は、圧縮レベルを指定する。デフォルトは 9 である。

```
with
gzip.open('somefile.gz', 'wt', compresslevel=5) as
f:
    f.write(text)
```

gzip.open() bz2.open() は、`encoding`、`errors`、`newline` を指定できる。

gzip.open() bz2.open() は、`encoding`、`errors`、`newline` を指定できる。

```
import gzip

f = open('somefile.gz', 'rb')
```

```
with
gzip.open(f, 'rt') as
g:
    text = g.read()
```

gzip bz2

## 5.8

### 5.8.1

### 5.8.2

iter() functools.partial()

```
from functools import
partial
RECORD_SIZE = 32
with
open('somefile.data', 'rb') as
```



```

f:
    records = iter(partial(f.read, RECORD_SIZE), b'')
    for
r in
records:
    ...

```

records is an iterator that yields records of size RECORD\_SIZE until it reaches the end of the file. The records are returned as bytes objects.

### 5.8.3 Iterators

The iter() function returns an iterator object that yields records of size RECORD\_SIZE until it reaches the end of the file. The records are returned as bytes objects.

The partial() function from the functools module returns a partial object that represents a function with some arguments fixed. In this case, the partial object represents a function that takes a file object as an argument and returns an iterator of records of size RECORD\_SIZE.

The iter() function is a built-in function that returns an iterator object. The partial() function is a function object that represents a function with some arguments fixed. The records are returned as bytes objects.

## 5.9 文件操作

### 5.9.1 文件

文件操作是Python中最基本的操作之一。在Python中，文件操作主要通过`open()`函数来实现。该函数返回一个文件对象，用于对文件进行读写操作。

### 5.9.2 文件对象

文件对象提供了多种方法来操作文件。其中，`readinto()`方法用于将文件内容读取到一个缓冲区中。该方法返回读取的字节数。

```
import os.path

def
read_into_buffer(filename):
    buf = bytearray(os.path.getsize(filename))
    with
open(filename, 'rb') as
f:
    f.readinto(buf)
    return
buf
```

文件操作是Python中最基本的操作之一。

```

>>> # Write a sample file

>>> with
open('sample.bin', 'wb') as
f:
...
f.write(b'Hello World')
...

>>> buf = read_into_buffer('sample.bin')
>>> buf
bytearray(b'Hello World')
>>> buf[0:5] = b'Hallo'
>>> buf
bytearray(b'Hallo World')
>>> with

open('newsample.bin', 'wb') as
f:
...
f.write(buf)
...

11
>>>

```

### 5.9.3 文件

readinto() 函数将文件内容读入一个 bytearray 对象（numpy 的 read() 函数使用 readinto() 函数实现）。

readinto() returns the number of bytes read into the buffer. If the buffer is not large enough to hold all the data, it will raise an IOError.

```
record_size = 32          # Size of each record (adjust value)

buf = bytearray(record_size)
with
open('somefile', 'rb') as
f:
while
True:
    n = f.readinto(buf)
    if
n < record_size:
    break

    # Use the contents of buf

    ...
```

memoryview() returns a memoryview object, which is a sequence of bytes that can be used to access the underlying memory. It is similar to a bytearray, but it is not mutable.

```
>>> buf
bytearray(b'Hello World')
```

```

>>> m1 = memoryview(buf)
>>> m2 = m1[-5:]
>>> m2
<memory at 0x100681390>
>>> m2[:] = b'WORLD'
>>> buf
bytearray(b'Hello WORLD')
>>>

```

f.readinto() 是 Python 中用于从文件对象 f 中读取数据并写入到缓冲区（bytearray 或 memoryview）的方法。

它返回一个整数，表示写入到缓冲区的数据字节数。

与 f.read() 不同，f.readinto() 不会创建新的字节串对象，而是直接将数据写入到指定的缓冲区中。这在处理大量数据时非常有用，因为它可以减少内存消耗。

在 Python 3.12 中，memoryview 对象支持更多的操作，包括从文件对象中读取数据。

## 5.10 文件操作

### 5.10.1 文件

在 Python 中，文件对象（File Object）用于与操作系统交互，读取和写入数据。

## 5.10.2 mmap

mmap module provides a way to map files or devices into memory. This allows you to access the data in the file as if it were in memory, which can be much faster than reading and writing the file repeatedly.

```
import os

import mmap

def
memory_map(filename, access=mmap.ACCESS_WRITE):
    size = os.path.getsize(filename)
    fd = os.open(filename, os.O_RDWR)
    return
mmap.mmap(fd, size, access=access)
```

The mmap module provides a way to map files or devices into memory. This allows you to access the data in the file as if it were in memory, which can be much faster than reading and writing the file repeatedly.

```
>>> size = 1000000
>>> with
open('data', 'wb') as
f:
...
f.seek(size-1)
...
f.write(b'\x00
```

```
' )  
...  
  
>>>
```

memory\_map() returns a memory map object

```
>>> m = memory_map('data')  
>>> len(m)  
1000000  
>>> m[0:10]  
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'  
>>> m[0]  
0  
>>> # Reassign a slice  
  
>>> m[0:11] = b'Hello World'  
>>> m.close()  
>>> # Verify that changes were made  
  
>>> with  
open('data', 'rb') as  
f:  
...     print  
(f.read(11))  
...  
  
b'Hello World'  
>>>
```

`mmap()` `mmap` `ACCESS_READ`  
`ACCESS_COPY`

```
>>> with
memory_map('data') as
m:
...     print
(len(m))
...     print
(m[0:10])
...

1000000
b'Hello World'
>>> m.closed
True
>>>
```

`memory_map()` `ACCESS_READ`  
`ACCESS_COPY`  
`access` `mmap.ACCESS_READ`

```
m = memory_map(filename, mmap.ACCESS_READ)
```

`mmap.ACCESS_COPY`



```
m = memory_map(filename, mmap.ACCESS_COPY)
```

## 5.10.3 mmap

mmap은 mmap 모듈을 사용하여 메모리 맵을 생성하고, seek(), read(), write() 메서드를 사용하여 메모리 맵을 읽고 쓰는 데 사용됩니다.

mmap() 메서드는 bytearray 객체와 memoryview 객체를 생성합니다.

```
>>> m = memory_map('data')
>>> # Memoryview of unsigned integers

>>> v = memoryview(m).cast('I')
>>> v[0] = 7
>>> m[0:4]
b'\x07\x00\x00\x00'
>>> m[0:4] = b'\x07\x01\x00\x00'

>>> v[0]
263
>>>
```

mmap은 메모리 맵을 생성하고, seek(), read(), write() 메서드를 사용하여 메모리 맵을 읽고 쓰는 데 사용됩니다.

Python 2.6 版本中，mmap 模块已经支持了 mmap 和 mmap2 两个子模块。mmap 模块是 mmap2 模块的超集，mmap2 模块是 mmap 模块的子集。mmap 模块是 mmap2 模块的超集，mmap2 模块是 mmap 模块的子集。

Python 的 mmap 模块提供了 mmap 和 mmap2 两个子模块。mmap 模块是 mmap2 模块的超集，mmap2 模块是 mmap 模块的子集。mmap 模块是 mmap2 模块的超集，mmap2 模块是 mmap 模块的子集。mmap 模块是 mmap2 模块的超集，mmap2 模块是 mmap 模块的子集。mmap 模块是 mmap2 模块的超集，mmap2 模块是 mmap 模块的子集。

Python 的 mmap 模块提供了 mmap 和 mmap2 两个子模块。mmap 模块是 mmap2 模块的超集，mmap2 模块是 mmap 模块的子集。mmap 模块是 mmap2 模块的超集，mmap2 模块是 mmap 模块的子集。mmap 模块是 mmap2 模块的超集，mmap2 模块是 mmap 模块的子集。mmap 模块是 mmap2 模块的超集，mmap2 模块是 mmap 模块的子集。mmap 模块是 mmap2 模块的超集，mmap2 模块是 mmap 模块的子集。

## 5.11 内存映射

### 5.11.1 内存映射

内存映射是一种将文件内容映射到内存的技术。它允许程序通过内存地址访问文件内容，而不需要通过文件系统。内存映射可以用于提高程序的性能，因为它可以减少文件 I/O 的次数。

### 5.11.2 内存映射

os.path module provides a portable way of representing file paths across different operating systems.

```
>>> import os

>>> path = '/Users/beazley/Data/data.csv'

>>> # Get the last component of the path

>>> os.path.basename(path)
'data.csv'
>>> # Get the directory name

>>> os.path.dirname(path)
'/Users/beazley/Data'

>>> # Join path components together

>>> os.path.join('tmp', 'data', os.path.basename(path))
'tmp/data/data.csv'

>>> # Expand the user's home directory

>>> path = '~/Data/data.csv'
>>> os.path.expanduser(path)
'/Users/beazley/Data/data.csv'

>>> # Split the file extension

>>> os.path.splitext(path)
('~/Data/data', '.csv')
>>>
```

## 5.11.3



os.path 模块提供了跨操作系统的函数，用于处理文件路径。os.path 模块在 UNIX 和 Windows 系统上都能使用。例如，Data/data.csv 在 UNIX 上是 Data/data.csv，而在 Windows 上是 Data\data.csv。

os.path 模块提供了跨操作系统的函数，用于处理文件路径。os.path 模块在 UNIX 和 Windows 系统上都能使用。

## 5.12 文件操作

### 5.12.1 文件

os.path 模块提供了跨操作系统的函数，用于处理文件路径。

### 5.12.2 目录

os.path 模块提供了跨操作系统的函数，用于处理文件路径。

```
>>> import os

>>> os.path.exists('/etc/passwd')
True
>>> os.path.exists('/tmp/spam')
False
>>>
```

os.path.isfile('/etc/passwd')  
False

```
>>> # Is a regular file

>>> os.path.isfile('/etc/passwd')
True

>>> # Is a directory

>>> os.path.isdir('/etc/passwd')
False

>>> # Is a symbolic link

>>> os.path.islink('/usr/local/bin/python3')
True

>>> # Get the file linked to

>>> os.path.realpath('/usr/local/bin/python3')
'/usr/local/bin/python3.3'
>>>
```

os.path

```
>>> os.path.getsize('/etc/passwd')
3669
>>> os.path.getmtime('/etc/passwd')
```

```
1272478234.0
>>> import time

>>> time.ctime(os.path.getmtime('/etc/passwd'))
'Wed Apr 28 13:10:34 2010'
>>>
```

## 5.12.3 `os.path`

`os.path` is a module that provides a portable way of using operating system path notation. It contains functions for manipulating file and directory paths and for determining whether a path refers to a file or directory, whether it exists, and whether it is a link.

```
>>> os.path.getsize('/Users/guido/Desktop/foo.txt')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/lib/python3.3/genericpath.py", line 49, in
getsize
    return
os.stat(filename).st_size
PermissionError: [Errno 13] Permission denied:
'/Users/guido/Desktop/foo.txt'
>>>
```

## 5.13 `os`

### 5.13.1 `os`

`os` is a module that provides a portable way of using operating system path notation. It contains functions for manipulating file and directory paths and for determining whether a path refers to a file or directory, whether it exists, and whether it is a link.

## 5.13.2 遍历目录

使用`os.listdir()`可以遍历指定目录，并返回一个包含该目录中所有文件和子目录名称的列表。

```
import os

names = os.listdir('somedir')
```

遍历目录时，我们通常只关心目录中的文件，而不关心子目录。因此，我们可以使用`os.path.isfile()`来判断一个文件是否是正则文件。

```
import os.path

# Get all regular files

names = [name for
name in
os.listdir('somedir')
if
os.path.isfile(os.path.join('somedir', name))]
# Get all dirs

dirnameames = [name for
name in
```

```
os.listdir('somedir')
    if
os.path.isdir(os.path.join('somedir', name))]
```

startswith()endswith()  
os.path.join()

```
pyfiles = [name for
name in
os.listdir('somedir')
    if
name.endswith('.py')]
```

globfnmatch  
os.path.join()

```
import glob

pyfiles = glob.glob('somedir/*.py')

from fnmatch import
fnmatch
pyfiles = [name for
name in
```



```
os.listdir('somedir')
    if
fnmatch(name, '*.py')]
```

## 5.13.3 文件

os.path 模块提供了与操作系统无关的路径操作函数。os.stat() 函数返回一个 stat 结构体，其中包含文件的元数据。

```
# Example of getting a directory listing
```

```
import os
```

```
import os.path
```

```
import glob
```

```
pyfiles = glob.glob('*.py')
```

```
# Get file sizes and modification dates
```

```
name_sz_date = [(name, os.path.getsize(name),
                    os.path.getmtime(name))
                 for
```

```
name in
```



## 5.14.1 文件

文件操作是Python中最基本的I/O操作之一。在Python中，文件操作是通过文件对象来完成的。文件对象提供了各种方法来读取和写入文件内容。

## 5.14.2 文件编码

在Python中，文件的编码是一个重要的概念。Python 3默认使用UTF-8编码来处理文本文件。你可以通过`sys.getfilesystemencoding()`来获取当前文件系统的默认编码。

```
>>> sys.getfilesystemencoding()
'utf-8'
>>>
```

在Python中，文件的编码是一个重要的概念。Python 3默认使用UTF-8编码来处理文本文件。你可以通过`sys.getfilesystemencoding()`来获取当前文件系统的默认编码。

```
>>> # Write a file using a unicode filename

>>> with
open('jalape\xfl
o.txt', 'w') as
f:
...
f.write('Spicy!')
...
```

```

6
>>> # Directory listing (decoded)

>>> import os

>>> os.listdir('.')
['jalapeño.txt']

>>> # Directory listing (raw)

>>> os.listdir(b'.')      # Note: byte string

[b'jalapen\xcc\x83o.txt']

>>> # Open file with raw filename

>>> with
open(b'jalapen\xcc\x83
o.txt') as
f:
...     print
(f.read())
...

Spicy!
>>>

```

os.listdir()
 open()

### 5.14.3 问题

在Python 3.6之前，字符串是不可变的。这意味着一旦你创建了一个字符串，你就不能改变它。这导致了一些问题，特别是当你在一个循环中需要多次修改同一个字符串时。Python 3.6引入了字符串的不可变性的限制，允许你在一个循环中多次修改同一个字符串。这被称为“字符串的不可变性”。

在Python 3.6之前，字符串是不可变的。这意味着一旦你创建了一个字符串，你就不能改变它。这导致了一些问题，特别是当你在一个循环中需要多次修改同一个字符串时。Python 3.6引入了字符串的不可变性的限制，允许你在一个循环中多次修改同一个字符串。这被称为“字符串的不可变性”。

在Python 5.15之前，字符串是不可变的。这意味着一旦你创建了一个字符串，你就不能改变它。这导致了一些问题，特别是当你在一个循环中需要多次修改同一个字符串时。Python 5.15引入了字符串的不可变性的限制，允许你在一个循环中多次修改同一个字符串。这被称为“字符串的不可变性”。

## 5.15 字符串的不可变性

### 5.15.1 问题

在Python 3.6之前，字符串是不可变的。这意味着一旦你创建了一个字符串，你就不能改变它。这导致了一些问题，特别是当你在一个循环中需要多次修改同一个字符串时。Python 3.6引入了字符串的不可变性的限制，允许你在一个循环中多次修改同一个字符串。这被称为“字符串的不可变性”。

### 5.15.2 解决方案

在Python 3.6之前，字符串是不可变的。这意味着一旦你创建了一个字符串，你就不能改变它。这导致了一些问题，特别是当你在一个循环中需要多次修改同一个字符串时。Python 3.6引入了字符串的不可变性的限制，允许你在一个循环中多次修改同一个字符串。这被称为“字符串的不可变性”。

```

def
bad_filename(filename):
    return

repr(filename)[1:-1]
try

:
    print

(filename)
except UnicodeEncodeError

:
    print

(bad_filename(filename))

```

## 5.15.3

Python
 `sys.getfilesystemencoding()`

`open()`

`os.listdir()`

Python

\\xhh surrogate encoding  
Unicode\\udchh  
bäd.txt Latin-1  
UTF-8

```
>>> import os

>>> files = os.listdir('.')
>>> files
['spam.py', 'b\\udce4d.txt', 'foo.txt']
>>>
```

open()  
\\udce4d.txt  
\\udce4d.txt

```
>>> for
name in
files:
...     print
(name)
...

spam.py
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
UnicodeEncodeError: 'utf-8' codec can't encode character
```

```
'\udce4' in
position 1: surrogates not allowed
>>>
```

0000000000\udce400000Unicode000000  
002000000000000000000000000000surrogate  
pair0000000000000000000000000000Unicode0000  
00  
00

```
>>> for
name in
files:
...     try
:
...
        print
(name)
...     except UnicodeEncodeError
:
...         print
(bad_filename(name))
...

spam.py
b\udce4d.txt
foo.txt
>>>
```



---

def bad\_filename(filename):  
 temp = filename.encode(sys.getfilesystemencoding(),  
errors='surrogateescape')  
 return  
temp.decode('latin-1')

```
def  
bad_filename(filename):  
    temp = filename.encode(sys.getfilesystemencoding(),  
errors='surrogateescape')  
    return  
temp.decode('latin-1')
```

for name in files:  
 try  
:  
 ...  
 print  
(name)  
 ...  
 except UnicodeEncodeError  
:

```
>>> for  
name in  
files:  
    ...  
        try  
:  
    ...  
        print  
(name)  
    ...  
    except UnicodeEncodeError  
:
```



```
import io
```

```
u = urllib.request.urlopen('http://www.python.org')
f = io.TextIOWrapper(u,encoding='utf-8')
text = f.read()
```

```
    process.detach()
    sys.stdout
```

```
>>> import sys
```

```
>>> sys.stdout.encoding
'UTF-8'
>>> sys.stdout = io.TextIOWrapper(sys.stdout.detach(),
encoding='latin-1')
>>> sys.stdout.encoding
'latin-1'
>>>
```

□ □

### 5.16.3 ☐☐

I/O

```

>>> f = open('sample.txt','w')
>>> f
<_io.TextIOWrapper name='sample.txt' mode='w' encoding='UTF-8'>
>>> f.buffer
<_io.BufferedWriter name='sample.txt'>
>>> f.buffer.raw
<_io.FileIO name='sample.txt' mode='wb'>
>>>

```

io.TextIOWrapper  
 io.BufferedWriter  
 I/O  
 io.FileIO  
 io.TextIOWrapper

io.TextIOWrapper  
 io.BufferedWriter  
 io.FileIO

```

>>> f
<_io.TextIOWrapper name='sample.txt' mode='w' encoding='UTF-8'>
>>> f = io.TextIOWrapper(f.buffer, encoding='latin-1')
>>> f
<_io.TextIOWrapper name='sample.txt' encoding='latin-1'>
>>> f.write('Hello')
Traceback (most recent call last):
  File "<stdin>", line 1, in
<module>
ValueError
: I/O operation on closed file.
>>>

```

打开文件f并写入内容  
关闭文件

detach()返回io.TextIOWrapper对象  
返回io.BufferedWriter对象  
返回io.BufferedReader对象

```
>>> f = open('sample.txt', 'w')
>>> f
<_io.TextIOWrapper name='sample.txt' mode='w' encoding='UTF-8'>
>>> b = f.detach()
>>> b
<_io.BufferedWriter name='sample.txt'>
>>> f.write('hello')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: underlying buffer has been detached
>>>
```

打开文件并写入内容

```
>>> f = io.TextIOWrapper(b, encoding='latin-1')
>>> f
<_io.TextIOWrapper name='sample.txt' encoding='latin-1'>
>>>
```

打开文件并写入内容  
返回文件对象  
返回文件对象

```
>>> sys.stdout = io.TextIOWrapper(sys.stdout.detach(),
encoding='ascii',
...
errors='xmlcharrefreplace')
>>> print
('Jalape\u00f1o')
Jalapeño
>>>
```

Python uses ASCII for the first 128 characters of the Unicode character set. Characters with ordinal values greater than 127 are encoded using a sequence of one or more bytes.

## 5.17 Unicode

### 5.17.1 Strings

Unicode strings are created by prefixing the string with 'u'.

### 5.17.2 Bytes

Bytes are created by prefixing the string with 'b'. The 'buffer' attribute of a file object returns a bytes object.

```
>>> import sys

>>> sys.stdout.write(b'Hello\n')
```

```
' )
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not bytes
>>> sys.stdout.buffer.write(b'Hello\n

' )
Hello
5
>>>
```

buffer

### 5.17.3

I/O  
 Unicode/buffer  
 /

sys.stdout  
 sys.stdout

## 5.18

### 5.18.1

Python 2.7.10 64-bit  
I/O 模块提供了与操作系统交互的接口。socket 模块提供了与网络交互的接口。  
Python 提供了与操作系统交互的接口。

## 5.18.2 文件操作

Python 提供了与操作系统交互的接口。open() 函数用于打开文件。Python 提供了与操作系统交互的接口。open() 函数用于打开文件。Python 提供了与操作系统交互的接口。open() 函数用于打开文件。

```
# Open a low-level file descriptor
```

```
import os
```

```
fd = os.open('somefile.txt', os.O_WRONLY | os.O_CREAT)  
# Turn into a proper file
```

```
f = open(fd, 'wt')  
f.write('hello world\n'  
)  
f.close()
```

Python 提供了与操作系统交互的接口。open() 函数用于打开文件。closefd=False



```
# Create a file object, but don't close underlying fd when done
```

```
f = open(fd, 'wt', closefd=False)
...
```

```

    UNIX
    I/O
    socket
    socket

```

```
from socket import

socket, AF_INET, SOCK_STREAM
def

echo_client(client_sock, addr):
    print

    ('Got connection from', addr)
        # Make text-mode file wrappers for socket reading/writing

        client_in = open(client_sock.fileno(), 'rt',
encoding='latin-1',
                                closefd=False)
        client_out = open(client_sock.fileno(), 'wt',
encoding='latin-1',
                                closefd=False)
        # Echo lines back to the client using file I/O
```

```

        for
line in
client_in:
    client_out.write(line)
    client_out.flush()
    client_sock.close()
def
echo_server(address):
    sock = socket(AF_INET, SOCK_STREAM)
    sock.bind(address)
    sock.listen(1)
    while
True:
        client, addr = sock.accept()
        echo_client(client, addr)

```

open()
 UNIX
 socket
 socket
 makefile()
 makefile()

stdout
 stdout

```

import sys

```

```
bstdout = open(sys.stdout.fileno(), 'wb', closefd=False)
bstdout.write(b'Hello World\n')
bstdout.flush()
```

A diagram consisting of a 6x25 grid of squares. The word "UNIX" is written in the 5th row, 3rd to 8th columns. The 6th row contains the word "SHELL" in the 1st to 7th columns.

## 5.19

## 5.19.1 ☐☐

## 5.19.2 ☐☐☐☐

tempfile module provides a convenient way to create temporary files.  
tempfile module provides a convenient way to create temporary files:  
tempfile module provides a convenient way to create temporary files:

```
from tempfile import
TemporaryFile
with
TemporaryFile('w+t') as
f:
    # Read/write to the file

    f.write('Hello World\n
')
    f.write('Testing\n
')
    # Seek back to beginning and read the data

    f.seek(0)
    data = f.read()
# Temporary file is destroyed
```

tempfile module provides a convenient way to create temporary files.

```
f = TemporaryFile('w+t')
# Use the temporary file

...
f.close()
# File is destroyed
```

`TemporaryFile()`은 기본적으로 `w+t` 모드로 열리고, `w+b` 모드로 열려면 `TemporaryFile(mode='w+b')`을 사용해야 합니다. `TemporaryFile()`은 `open()`과 유사하게 사용됩니다.

```
with
TemporaryFile('w+t', encoding='utf-8', errors='ignore') as
f:
    ...
```

UNIX에서는 `TemporaryFile()` 대신 `NamedTemporaryFile()`을 사용하여 임시 파일을 생성할 수 있습니다.

```
from tempfile import
NamedTemporaryFile
with
NamedTemporaryFile('w+t') as
f:
    print
    ('filename is:', f.name)

    ...
# File automatically destroyed
```

```
f.name
TemporaryFile(delete=False)
```

```
with
NamedTemporaryFile('w+t', delete=False) as
f:
    print
('filename is:', f.name)
...
```

```
tempfile.TemporaryDirectory()
```

```
from tempfile import
TemporaryDirectory
with
TemporaryDirectory() as
dirname:
    print
```

```

('dirname is:', dirname)

# Use the directory

...
# Directory and all contents destroyed

```

## 5.19.3 `tempfile`

`tempfile` module provides functions to create temporary files and directories. The functions are `TemporaryFile()`, `NamedTemporaryFile()`, `TemporaryDirectory()`, `mkstemp()`, and `mkdtemp()`.

```

>>> import tempfile

>>> tempfile.mkstemp()
(3, '/var/folders/7W/7WZl5sfZEF0pljrEB1UMWE+++TI/-Ttmp-/tmp7fefhv')
>>> tempfile.mkdtemp()
'/var/folders/7W/7WZl5sfZEF0pljrEB1UMWE+++TI/-Ttmp-/tmp5wvcv6'
>>>

```

mkstemp() 函数在系统上创建一个临时文件，并返回该文件的描述符。该函数在系统上创建一个临时文件，并返回该文件的描述符。该函数在系统上创建一个临时文件，并返回该文件的描述符。

tempfile.gettempdir() 函数返回系统上的临时目录。该函数返回系统上的临时目录。该函数返回系统上的临时目录。

```
>>> tempfile.gettempdir()
'/var/folders/7W/7WZl5sfZEF0pljrEB1UMWE+++TI/-Tmp- '
>>>
```

tempfile.NamedTemporaryFile(prefix, suffix, dir) 函数在系统上创建一个临时文件，并返回该文件的描述符。该函数在系统上创建一个临时文件，并返回该文件的描述符。该函数在系统上创建一个临时文件，并返回该文件的描述符。

```
>>> f = NamedTemporaryFile(prefix='mytemp', suffix='.txt',
dir='/tmp')
>>> f.name
'/tmp/mytemp8ee899.txt'
>>>
```

tempfile.TemporaryFile() 函数在系统上创建一个临时文件，并返回该文件的描述符。该函数在系统上创建一个临时文件，并返回该文件的描述符。该函数在系统上创建一个临时文件，并返回该文件的描述符。



<http://docs.python.org/3/library/tempfile.html>

## 5.20 環境構築

### 5.20.1 準備

環境構築の準備として、Python 3.6.0 以上をインストールしてください。また、PySerial をインストールしてください。

### 5.20.2 環境構築

環境構築の準備として、Python の I/O モジュールである PySerial をインストールしてください。PySerial は、Python でシリアルポートを制御するためのモジュールです。

```
import serial
```

```
ser = serial.Serial('/dev/tty.usbmodem641', # Device name  
varies
```

```
    baudrate=9600,  
    bytesize=8,  
    parity='N',  
    stopbits=1)
```

Windows 0 1 read() readline() write()

```
ser.write(b'G1 X50 Y50\r\n')
resp = ser.readline()
```

### 5.20.3

pySerial RTS-CTS Serial() rtscts=True pySerial

I/O / struct

## 5.21 Python

## 5.21.1 序列化

序列化是指将Python对象转换为字节流的过程。这个过程通常用于将对象保存到文件或通过网络传输。

## 5.21.2 使用pickle

Python的pickle模块提供了一种简单的方法，可以将Python对象序列化为字节流。以下是一个使用pickle模块的示例。

```
import pickle

data = ... # Some Python object

f = open('somefile', 'wb')
pickle.dump(data, f)
```

上述代码将对象data序列化并保存到文件somefile中。pickle.dump()方法用于将对象序列化并写入文件。

```
s = pickle.dumps(data)
```

上述代码将对象data序列化为字节流s。pickle.dumps()方法用于将对象序列化为字节流。pickle.loads()方法用于从字节流中加载对象。

```
# Restore from a file
```

```
f = open('somefile', 'rb')
data = pickle.load(f)
# Restore from a string

data = pickle.loads(s)
```

## 5.21.3 序列化

序列化是指将Python对象转换为字节流的过程。这个过程通常使用pickle模块的dump()和load()函数来完成。序列化后的数据可以存储在文件中，也可以通过网络传输。反序列化是指将字节流转换回Python对象的过程，通常使用pickle模块的loads()函数来完成。

pickle模块是Python标准库的一部分，它提供了一种简单的方法来实现对象的序列化。使用pickle模块，可以将任何Python对象（包括列表、字典、元组、字符串等）转换为字节流，并将该字节流写入文件。同样，也可以从文件中读取字节流，并将其转换回原始的Python对象。

```
>>> import pickle

>>> f = open('somedata', 'wb')
>>> pickle.dump([1, 2, 3, 4], f)
>>> pickle.dump('hello', f)
>>> pickle.dump({'Apple', 'Pear', 'Banana'}, f)
>>> f.close()
>>> f = open('somedata', 'rb')
>>> pickle.load(f)
[1, 2, 3, 4]
>>> pickle.load(f)
'hello'
```

```
>>> pickle.load(f)
{'Apple', 'Pear', 'Banana'}
>>>
```

pickle module  
 pickle module

```
>>> import math

>>> import pickle.

>>> pickle.dumps(math.cos)
b'\x80\x03cmath\ncos\nq\x00.'
```

Python  
 Python



Python

pickle.load()  
 pickle module  
 pickle module  
 Python

`pickle` 模块提供了将任意对象序列化为字节流（序列化）以及从字节流中恢复对象（反序列化）的功能。

在 Python 中，使用 `pickle` 模块进行序列化和反序列化。序列化是将对象转换为字节流的过程，而反序列化则是将字节流转换回对象的过程。序列化后的对象可以存储在文件中，也可以通过网络传输。反序列化时，需要从文件中读取字节流，并将其转换回原始对象。

序列化过程通常涉及调用 `pickle.dump()` 或 `pickle.dumps()` 方法，这些方法将对象写入文件或返回字节流。反序列化过程则涉及调用 `pickle.load()` 或 `pickle.loads()` 方法，这些方法从文件或字节流中读取数据并返回对象。此外，`pickle` 模块还提供了 `__getstate__()` 和 `__setstate__()` 方法，用于自定义对象的序列化行为。

```
# countdown.py

import time

import threading

class Countdown:
    :
    def
__init__(self, n):

self.n = n

self.thr = threading.Thread(target=self.run)
```

```
self.thr.daemon = True
```

```
self.thr.start()
```

```
def
```

```
run(self):
```

```
    while
```

```
self.n > 0:
```

```
    print
```

```
('T-minus', self.n)
```

```
self.n -= 1
```

```
time.sleep(5)
```

```
def
```

```
__getstate__(self):
```

```
    return
```

```
self.n
```

```
def
```

```
__setstate__(self, n):
```

```
self.__init__(n)
```

□□□□□□□□□□pickle□□□

```
>>> import countdown
```





HDF5

pickle Python  
 pickle  
 XML CSV  
 JSON

pickle  
 <http://docs.python.org/3/library/pickle.html>

## 6 数据格式

Python 支持多种数据格式，包括 CSV、JSON、XML 等。本章将介绍如何使用 Python 处理这些数据格式。

### 6.1 CSV 文件

#### 6.1.1 简介

CSV 文件是一种常见的数据格式，用于存储表格数据。

#### 6.1.2 读取 CSV 文件

使用 Python 的 `csv` 模块可以方便地读取 CSV 文件。以下是一个示例，展示了如何读取 `stocks.csv` 文件。

```
Symbol,Price,Date,Time,Change,Volume
"AA",39.48,"6/11/2007","9:36am",-0.18,181800
"AIG",71.38,"6/11/2007","9:36am",-0.15,195500
"AXP",62.58,"6/11/2007","9:36am",-0.46,935000
"BA",98.31,"6/11/2007","9:36am",+0.12,104800
"C",53.08,"6/11/2007","9:36am",-0.25,360900
"CAT",78.29,"6/11/2007","9:36am",-0.23,225400
```

```
import csv

with
open('stocks.csv') as
f:
    f_csv = csv.reader(f)
    headers = next(f_csv)
    for
row in
f_csv:
    # Process row

    ...
```

```
from collections import  
namedtuple  
with  
open('stock.csv') as
```

```

f:
    f_csv = csv.reader(f)
    headings = next(f_csv)
    Row = namedtuple('Row', headings)
    for
r in
f_csv:
    row = Row(*r)
    # Process row

    ...

```

row.Symbol  
 row.Change  
 Python

```

import csv

with
open('stocks.csv') as
f:
    f_csv = csv.DictReader(f)
    for
row in
f_csv:

```

```
# process row
```

```
...
```

```
row['Symbol'] * row['Change']
```

```
CSV(csv.writer(f))
```

```
headers = ['Symbol', 'Price', 'Date', 'Time', 'Change', 'Volume']
rows = [('AA', 39.48, '6/11/2007', '9:36am', -0.18, 181800),
        ('AIG', 71.38, '6/11/2007', '9:36am', -0.15, 195500),
        ('AXP', 62.58, '6/11/2007', '9:36am', -0.46, 935000),
        ]
with
open('stocks.csv', 'w') as
f:
    f_csv = csv.writer(f)
    f_csv.writerow(headers)
    f_csv.writerows(rows)
```

```
headers = ['Symbol', 'Price', 'Date', 'Time', 'Change',
'Volume']
rows = [{'Symbol': 'AA', 'Price': 39.48, 'Date': '6/11/2007',
'Time': '9:36am', 'Change': -0.18, 'Volume': 181800},
{'Symbol': 'AIG', 'Price': 71.38, 'Date': '6/11/2007',
'Time': '9:36am', 'Change': -0.15, 'Volume': 195500},
{'Symbol': 'AXP', 'Price': 62.58, 'Date': '6/11/2007',
'Time': '9:36am', 'Change': -0.46, 'Volume': 935000},
```

```

    ]
with
open('stocks.csv', 'w') as
f:
    f_csv = csv.DictWriter(f, headers)
    f_csv.writeheader()
    f_csv.writerows(rows)

```

## 6.1.3

CSV CSV

```

with
open('stocks.csv') as
f:
    for
line in
f:
    row = line.split(',')
    # process row

    ...

```

csv 모듈은 Excel과 CSV 파일을 읽고 쓰는 데 사용됩니다.  
csv 모듈은 CSV 파일을 읽고 쓰는 데 사용됩니다.

csv 모듈은 Excel과 CSV 파일을 읽고 쓰는 데 사용됩니다.  
csv 모듈은 CSV 파일을 읽고 쓰는 데 사용됩니다.  
csv 모듈은 CSV 파일을 읽고 쓰는 데 사용됩니다.  
csv 모듈은 CSV 파일을 읽고 쓰는 데 사용됩니다.  
csv 모듈은 CSV 파일을 읽고 쓰는 데 사용됩니다.

```
# Example of reading tab-separated values
```

```
with
```

```
open('stock.tsv') as
```

```
f:
```

```
    f_tsv = csv.reader(f, delimiter='\t
```

```
')
```

```
    for
```

```
row in
```

```
f_tsv:
```

```
    # Process row
```

```
    ...
```

csv 모듈은 CSV 파일을 읽고 쓰는 데 사용됩니다.  
csv 모듈은 CSV 파일을 읽고 쓰는 데 사용됩니다.

ValueError [1]

Street Address,Num-Premises,Latitude,Longitude 5412 N CLARK,10,41.980262,-87.668452
--

ValueError

```
import re

with
open('stock.csv') as
f:
    f_csv = csv.reader(f)
    headers = [ re.sub('[^a-zA-Z_]', '_', h) for
h in
next(f_csv) ]
    Row = namedtuple('Row', headers)
    for
r in
f_csv:
    row = Row(*r)
    # Process row

    ...
```



#####CSV#####  
#####  
#####CSV#####

```
col_types = [str, float, str, str, float, int]
with
open('stocks.csv') as
f:
    f_csv = csv.reader(f)
    headers = next(f_csv)
    for
row in
f_csv:
    # Apply conversions to the row items

    row = tuple(convert(value) for
convert, value in
zip(col_types, row))
    ...
```

#####

```
print
('Reading as dicts with type conversion')
field_types = [ ('Price', float),
                ('Change', float),
                ('Volume', int) ]
with
```

```

open('stocks.csv') as
f:
    for
row in
csv.DictReader(f):
    row.update((key, conversion(row[key]))
               for
key, conversion in
field_types)
    print
(row)

```

1. CSV 파일을 Python으로 읽어들이기

2. CSV 파일을 Pandas로 읽어들이기

Pandas는 Python에서 데이터를 다루는 데 사용되는 라이브러리입니다. Pandas를 사용하기 위해서는 <http://pandas.pydata.org>에서 Pandas를 설치해야 합니다.

Pandas의 `read_csv()` 함수를 사용하여 CSV 파일을 DataFrame으로 읽어들이는 방법은 다음과 같습니다.

6.13 Pandas를 사용하여 CSV 파일을 읽어들이기

## 6.2 JSON

## 6.2.1 JSON

JSON (JavaScript Object Notation) is a lightweight data interchange format. It is easy for humans to read and write, and easy for machines to parse and generate.

## 6.2.2 JSON in Python

Python has a built-in module called `json` that can be used to work with JSON data. The `json` module provides two main functions: `json.dumps()` to convert a Python object into a JSON string, and `json.loads()` to convert a JSON string back into a Python object. This is similar to how the `pickle` module works in Python, but JSON is a standard format that can be used across different programming languages.

```
import json

data = {
    'name' : 'ACME',
    'shares' : 100,
    'price' : 542.23
}
json_str = json.dumps(data)
```

Python also provides a way to load JSON data from a string. The `json.loads()` function can be used to convert a JSON string back into a Python object.

```
data = json.loads(json_str)
```

python json.dump() json.load() JSON

```
# Writing JSON data

with
open('data.json', 'w') as
f:
    json.dump(data, f)
# Reading data back

with
open('data.json', 'r') as
f:
    data = json.load(f)
```

### 6.2.3

JSON None bool int float str  
JSON key Python Web

JSON和Python的对应关系  
True对应true False对应false  
None对应null

```
>>> json.dumps(False)
'false'
>>> d = {'a': True,
...
'b': 'Hello',
...
'c': None}
>>> json.dumps(d)
'{"b": "Hello", "c": null, "a": true}'
>>>
```

使用JSON模块——使用pprint模块  
pprint()模块  
Twitter

```
>>> from urllib.request import
urlopen
>>> import json

>>> u = urlopen('http://search.twitter.com/search.json?
q=python&rpp=5')
>>> resp = json.loads(u.read().decode('utf-8'))
>>> from pprint import
```

```

pprint
>>> pprint(resp)
{'completed_in': 0.074,
 'max_id': 264043230692245504,
 'max_id_str': '264043230692245504',
 'next_page': '?
page=2&max_id=264043230692245504&q=python&rpp=5',
 'page': 1,
 'query': 'python',
 'refresh_url': '?since_id=264043230692245504&q=python',
 'results': [{'created_at': 'Thu, 01 Nov 2012 16:36:26 +0000',
               'from_user': ...
             },
             {'created_at': 'Thu, 01 Nov 2012 16:36:14 +0000',
               'from_user': ...
             },
             {'created_at': 'Thu, 01 Nov 2012 16:36:13 +0000',
               'from_user': ...
             },
             {'created_at': 'Thu, 01 Nov 2012 16:36:07 +0000',
               'from_user': ...
             },
             {'created_at': 'Thu, 01 Nov 2012 16:36:04 +0000',
               'from_user': ...
             }],
 'results_per_page': 5,
 'since_id': 0,
 'since_id_str': '0'}
>>>

```

Python 2.7.3 的 JSON 模块提供了 json.loads() 函数，
 它返回一个 object\_pairs\_hook 参数，默认为 object\_hook。
 如果你希望返回一个 OrderedDict，你可以传递 OrderedDict 类。
 如果你希望返回一个普通字典，你可以传递 dict 类。

```

>>> s = '{"name": "ACME", "shares": 50, "price": 490.1}'
>>> from collections import

OrderedDict
>>> data = json.loads(s, object_pairs_hook=OrderedDict)
>>> data
OrderedDict([('name', 'ACME'), ('shares', 50), ('price',
490.1)])
>>>

```

## JSONとPython

```

>>> class JSONObject
:
...     def
__init__(self, d):
...
self.__dict__ = d
...

>>>
>>> data = json.loads(s, object_hook=JSONObject)
>>> data.name
'ACME'
>>> data.shares
50
>>> data.price
490.1
>>>

```

JSONとPython  
 \_\_init\_\_()

□□□□□□□□□□□□

□□□□□□□□JSON□□□□□□□□□□□□□□□□□□  
□□□□□□□json.dumps()□□□□□indent□□□□□□□  
□□□□□pprint()□□□□□□□□□□□□□□□□□□□□□□

```
>>> print
(json.dumps(data))
{"price": 542.23, "name": "ACME", "shares": 100}
>>> print
(json.dumps(data, indent=4))
{
    "price": 542.23,
    "name": "ACME",
    "shares": 100
}
>>>
```

□□□□□□□□□□□□□□□□□□□□□□□□sort\_keys□□□

```
>>> print
(json.dumps(data, sort_keys=True))
{"name": "ACME", "price": 542.23, "shares": 100}
>>>
```

□□□□□□□□□□□□JSON□□□□□□□

```
>>> class Point
```





**def**

```
serialize_instance(obj):  
    d = { '__classname__' : type(obj).__name__ }  
    d.update(vars(obj))  
    return d
```

□□□□□□□□□□□□□□□□□□□□□□□□

*# Dictionary mapping names to known classes*

```
classes = {  
    'Point' : Point  
}
```

**def**

```
unserialize_object(d):  
    clsname = d.pop('__classname__', None)  
    if
```

```
clsname:  
        cls = classes[clsname]  
        obj = cls.__new__(cls) # Make instance without calling  
__init__
```

**for**

key, value **in**

```
d.items():  
        setattr(obj, key, value)  
    return
```

obj

**else**



## 6.3.2 爬取

xml.etree.ElementTree 是 Python 标准库中用于处理 XML 文档的模块。Planet Python 的 RSS 订阅地址是 <http://planet.python.org>。我们将使用这个地址来爬取 RSS 订阅信息。

```
from urllib.request import
urlopen
from xml.etree.ElementTree import
parse
# Download the RSS feed and parse it

u = urlopen('http://planet.python.org/rss20.xml')
doc = parse(u)
# Extract and output tags of interest

for
item in
doc.iterfind('channel/item'):
    title = item.findtext('title')
    date = item.findtext('pubDate')
    link = item.findtext('link')

    print
(title)
    print
(date)
    print
(link)
```

( )

[illegible]

```
Steve Holden: Python for Data Analysis
Mon, 19 Nov 2012 02:13:51 +0000
http://holdenweb.blogspot.com/2012/11/python-for-data-
analysis.html
Vasudev Ram: The Python Data model (for v2 and v3)
Sun, 18 Nov 2012 22:06:47 +0000
http://jugad2.blogspot.com/2012/11/the-python-data-model.html
Python Diary: Been playing around with Object Databases
Sun, 18 Nov 2012 20:40:29 +0000
http://www.pythondiary.com/blog/Nov.18,2012/been-...-object-
databases.html
Vasudev Ram: Wakari, Scientific Python in the cloud
Sun, 18 Nov 2012 20:19:41 +0000
http://jugad2.blogspot.com/2012/11/wakari-scientific-python-
in-cloud.html
Jesse Jiryu Davis: Toro: synchronization primitives for
Tornado coroutines
Sun, 18 Nov 2012 20:17:49 +0000
http://feedproxy.google.com/~r/EmptysquarePython/~3/_D0ZT2Kd0h
0/
```

```
print()
```

### 6.3.3 □□

XML RSS XML

XML RSS XML

```
<?xml version="1.0"?>
<rss version="2.0"
xmlns:dc="http://purl.org/dc/elements/1.1/">
<channel>
  <title>Planet Python</title>
  <link>http://planet.python.org/</link>
  <language>en</language>
  <description>Planet Python -
http://planet.python.org/</description>
  <item>
    <title>Steve Holden: Python for Data Analysis</title>
    <guid>http://holdenweb.blogspot.com/...-data-
analysis.html</guid>
    <link>http://holdenweb.blogspot.com/...-data-
analysis.html</link>
    <description>...</description>
    <pubDate>Mon, 19 Nov 2012 02:13:51 +0000</pubDate>
  </item>
  <item>
    <title>Vasudev Ram: The Python Data model (for v2 and v3)
</title>
    <guid>http://jugad2.blogspot.com/...-data-
model.html</guid>
    <link>http://jugad2.blogspot.com/...-data-
model.html</link>
    <description>...</description>
    <pubDate>Sun, 18 Nov 2012 22:06:47 +0000</pubDate>
  </item>
  <item>
    <title>Python Diary: Been playing around with Object
Databases</title>
    <guid>http://www.pythondiary.com/...-object-
databases.html</guid>
```

```

    <link>http://www.pythondiary.com/...-object-
databases.html</link>
    <description>...</description>
    <pubDate>Sun, 18 Nov 2012 20:40:29 +0000</pubDate>
  </item>
  ...
</channel>
</rss>

```

xml.etree.ElementTree.parse() 解析 XML  
 XML 文档并返回一个 ElementTree 对象  
 find() 方法用于在 XML 文档中查找指定的元素  
 iterfind() 方法用于在 XML 文档中查找指定的元素  
 findtext() 方法用于在 XML 文档中查找指定的元素的文本内容  
 channel/item 元素

解析 XML 文档并返回一个 ElementTree 对象  
 在 XML 文档中查找指定的元素  
 doc.iterfind('channel/item') 返回一个迭代器  
 遍历迭代器中的元素  
 “channel” 元素  
 “item” 元素  
 doc 对象  
 “rss” 元素  
 item.findtext() 方法  
 “item” 元素

ElementTree 对象  
 tag 属性  
 text 属性  
 get() 方法  
 返回元素的文本内容

```

>>> doc
<xml.etree.ElementTree.ElementTree object at 0x101339510>
>>> e = doc.find('channel/title')
>>> e
<Element 'title' at 0x10135b310>

```

```
>>> e.tag
'title'
>>> e.text
'Planet Python'
>>> e.get('some_attribute')
>>>
```

xml.etree.ElementTree  
XML  
lxml  
ElementTree  
lxml  
from  
lxml.etree import parse  
XML  
lxml  
XSLT  
XPath

## 6.4 XML

### 6.4.1

XML

### 6.4.2

XML



```

from xml.etree.ElementTree import
iterparse
def
parse_and_remove(filename, path):
    path_parts = path.split('/')
    doc = iterparse(filename, ('start', 'end'))
    # Skip the root element

    next(doc)
    tag_stack = []
    elem_stack = []
    for
event, elem in
doc:
    if
event == 'start':
        tag_stack.append(elem.tag)
        elem_stack.append(elem)
    elif
event == 'end':
        if
tag_stack == path_parts:
            yield
elem
            elem_stack[-2].remove(elem)
        try
:
            tag_stack.pop()
            elem_stack.pop()
        except IndexError
:
            pass

```



```

        <type_of_service_request>Pot Hole in
Street</type_of_service_request>
        <current_activity>Final Outcome</current_activity>
        <most_recent_action>CDOT Street Cut ...
Outcome</most_recent_action>
        <street_address>3510 W NORTH AVE</street_address>
        <zip>60647</zip>
        <x_coordinate>1152732.14127696</x_coordinate>
        <y_coordinate>1910409.38979075</y_coordinate>
        <ward>26</ward>
        <police_district>14</police_district>
        <community_area>23</community_area>
        <latitude>41.91002084292946</latitude>
        <longitude>-87.71435952353961</longitude>
        <location latitude="41.91002084292946"
                                longitude="-87.71435952353961" />

    </row>
</row>
</response>

```

ZIP  
 code

```

from xml.etree.ElementTree import
parse
from collections import
Counter
potholes_by_zip = Counter()
doc = parse('potholes.xml')
for
    pothole in
        doc.iterfind('row/row'):
            potholes_by_zip[pothole.findtext('zip')] += 1
for
    zipcode, num in

```

```
potholes_by_zip.most_common():
    print
(zipcode, num)
```

00000000000000000000XML000000000000  
 0000000000000000000000000000000000450 MB00000000  
 0000000000000000000000000000000000

```
from collections import
Counter
potholes_by_zip = Counter()
data = parse_and_remove('potholes.xml', 'row/row')
for
    pothole in
        data:
            potholes_by_zip[pothole.findtext('zip')] += 1
    for
        zipcode, num in
            potholes_by_zip.most_common():
                print
                    (zipcode, num)
```

7 MB

### 6.4.3 □□

ElementTree  
iterparse()  
XML  
1  
start/end  
start-ns/end-ns  
iterparse()  
event  
elem  
event  
elem  
XML

```
>>> data = iterparse('potholes.xml', ('start', 'end'))
>>> next(data)
('start', <Element 'response' at 0x100771d60>)
>>> next(data)
('start', <Element 'row' at 0x100771e68>)
>>> next(data)
('start', <Element 'row' at 0x100771fc8>)
>>> next(data)
('start', <Element 'creation_date' at 0x100771f18>)
>>> next(data)
('end', <Element 'creation_date' at 0x100771f18>)
>>> next(data)
('start', <Element 'status' at 0x1006a7f18>)
>>> next(data)
('end', <Element 'status' at 0x1006a7f18>)
>>>
```

start  
end  
start-ns  
end-ns  
XML

start  
end  
current  
hierarchical

`parse_and_remove()` 函数返回一个生成器  
`yield` 语句

函数 `yield` 返回一个 `ElementTree` 对象  
和根元素

```
elem_stack[-2].remove(elem)
```

函数 `yield` 返回一个生成器  
返回一个生成器，用于遍历 XML 文档  
的所有元素

函数 `yield` 返回一个生成器  
返回一个生成器，用于遍历 XML 文档  
的所有元素

函数 `yield` 返回一个生成器  
返回一个生成器，用于遍历 XML 文档  
的所有元素

## 6.5 遍历 XML

### 6.5.1 遍历

## PythonでXML

### 6.5.2 XML

`xml.etree.ElementTree`でXMLを  
簡単に扱うことができる。

```
from xml.etree.ElementTree import
Element
def
dict_to_xml(tag, d):

    Turn a simple dict of key/value pairs into XML

    ...

    elem = Element(tag)
    for
key, val in
d.items():
        child = Element(key)
        child.text = str(val)
        elem.append(child)
    return
elem
```

字典转xml

```
>>> s = { 'name': 'G00G', 'shares': 100, 'price':490.1 }
>>> e = dict_to_xml('stock', s)
>>> e
<Element 'stock' at 0x1004b64c8>
>>>
```

字典转xml之Element与xml.etree.ElementTree的toString()方法

```
>>> from xml.etree.ElementTree import
toString
>>> toString(e)
b'<stock><price>490.1</price><shares>100</shares>
<name>G00G</name>'
>>>
```

字典转xml之Element的set()方法

```
>>> e.set('_id','1234')
>>> toString(e)
b'<stock _id="1234"><price>490.1</price><shares>100</shares>
<name>G00G</name>
</stock>'
>>>
```

字典转xml之OrderedDict



## 6.5.3 字典

字典XML字符串

```
def
dict_to_xml_str(tag, d):
    '''
        Turn a simple dict of key/value pairs into XML
    '''

    parts = ['<{}>'.format(tag)]
    for
key, val in
d.items():
    parts.append('<{0}>{1}</{0}>'.format(key, val))
    parts.append('</{}>'.format(tag))
    return

''.join(parts)
```

字典XML字符串

```
>>> d = { 'name' : '<spam>' }
>>> # String creation

>>> dict_to_xml_str('item',d)
```

```
'<item><name><spam></name></item>'
>>> # Proper XML creation

>>> e = dict_to_xml('item',d)
>>> tostring(e)
b'<item><name><spam></name></item>'
>>>
```

<>  &lt; &gt;

```
xml.sax.saxutils.escape()unescape()

```

```
>>> from xml.sax.saxutils import
escape, unescape
>>> escape('<spam>')
'<spam>'
>>> unescape(_)
'<spam>'
>>>
```

```

    Element
    Element
    XML
    XML

```

## 6.6 用XML

### 6.6.1

XMLXML

### 6.6.2

xml.etree.ElementTree  
pred.xml

```
<?xml version="1.0"?>
<stop>

  <id>
14791</id>

  <nm>
Clark &
Balmoral</nm>

  <sri>

    <rt>
22</rt>
```

<d>

North Bound</d>

<dd>

North Bound</dd>

</sri>

<cr>

22</cr>

<pre>

<pt>

5 MIN</pt>

<fd>

Howard</fd>

<v>

1378</v>

<rn>

22</rn>

</pre>

```
<pre>

    <pt>
15 MIN</pt>

    <fd>
Howard</fd>

    <v>
1867</v>

    <rn>
22</rn>

</pre>

</stop>
```

ElementTree

```
>>> from xml.etree.ElementTree import
parse, Element
>>> doc = parse('pred.xml')
>>> root = doc.getroot()
>>> root
<Element 'stop' at 0x100770cb0>
```

```
>>> # Remove a few elements

>>> root.remove(root.find('sri'))
>>> root.remove(root.find('cr'))
>>> # Insert a new element after <nm>...</nm>

>>> root.getchildren().index(root.find('nm'))
1
>>> e = Element('spam')
>>> e.text = 'This is a test'
>>> root.insert(2, e)
>>> # Write back to a file

>>> doc.write('newpred.xml', xml_declaration=True)
>>>
```

[illegible]

```
<?xml version='1.0' encoding='us-ascii'?>
<stop>

    <id>
14791</id>

    <nm>
Clark &
Balmoral</nm>

    <spam>
This is a test</spam><pre>
```

<pt>

5 MIN</pt>

<fd>

Howard</fd>

<v>

1378</v>

<rn>

22</rn>

</pre>

<pre>

<pt>

15 MIN</pt>

<fd>

Howard</fd>

<v>

1867</v>

<rn>

22</rn>

```
</pre>
```

```
</stop>
```

## 6.6.3 列表

XML 列表是 XML 文档中元素的一个有序集合。列表中的元素可以通过索引访问，类似于 Python 中的列表。列表支持 `remove()`、`insert()`、`append()` 等方法。列表的索引从 0 开始，`element[i]` 表示列表中的第 `i` 个元素，`element[i:j]` 表示从第 `i` 个元素到第 `j` 个元素的子列表。

在 6.5 节中，我们介绍了 `Element` 类，它是 XML 列表的基本单元。在本节中，我们将进一步探讨 XML 列表的操作。

## 6.7 使用 XML 列表

### 6.7.1 列表

XML 列表是 XML 文档中元素的一个有序集合。列表中的元素可以通过索引访问，类似于 Python 中的列表。

### 6.7.2 列表操作



# XML

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<top>
```

```
<author>
```

```
David Beazley</author>
```

```
<content>
```

```
<html
```

```
xmlns="http://www.w3.org/1999/xhtml">
```

```
<head>
```

```
<title>
```

```
Hello World</title>
```

```
</head>
```

```
<body>
```

```
<h1>
```

```
Hello World!</h1>
```

```
</body>
```

```
</html>
```

**</content>**

**</top>**

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> # Some queries that work
```

```
>>> doc.findtext('author')  
'David Beazley'
```

```
>>> doc.find('content')  
<Element 'content' at 0x100776ec0>
```

```
>>> # A query involving a namespace (doesn't work)
```

```
>>> doc.find('content/html')
```

```
>>> # Works if fully qualified
```

```
>>> doc.find('content/{http://www.w3.org/1999/xhtml}html')  
<Element '{http://www.w3.org/1999/xhtml}html' at 0x1007767e0>
```

```
>>> # Doesn't work
```

```
>>>
```

```
doc.findtext('content/{http://www.w3.org/1999/xhtml}html/head/  
title')
```

```
>>> # Fully qualified
```

```
>>> doc.findtext('content/{http://www.w3.org/1999/xhtml}html/'  
...)
```

```
'{http://www.w3.org/1999/xhtml}head/{http://www.w3.org/1999/xh
```

```
tml}title')
'Hello World'
>>>
```

XX  
XX

```
class XMLNamespaces
:
    def
__init__(self, **kwargs):

self.namespaces = {}
    for
name, uri in
kwargs.items():

self.register(name, uri)
    def
register(self, name, uri):

self.namespaces[name] = '{'+uri+'}'
    def
__call__(self, path):
    return
path.format_map(self.namespaces)
```

```
>>> ns = XMLNamespaces(html='http://www.w3.org/1999/xhtml')
>>> doc.find(ns('content/{html}html'))
<Element '{http://www.w3.org/1999/xhtml}html' at 0x1007767e0>
>>>
doc.findtext(ns('content/{html}html/{html}head/{html}title'))
'Hello World'
>>>
```

```

    namespace XML
    XMLNamespaces
    URI

```

```

ElementTree
iterparse()

```

```
>>> from xml.etree.ElementTree import
iterparse
>>> for
evt, elem in
iterparse('ns2.xml', ('end', 'start-ns', 'end-ns')):
...     print
(evt, elem)
...
```



## 6.8.2 数据库

Python数据库API

```
stocks = [  
  
    ('GOOG', 100, 490.1),  
    ('AAPL', 50, 545.75),  
    ('FB', 150, 7.45),  
    ('HPQ', 75, 33.2),  
]
```

Python数据库API  
PEP 249数据库API  
SQL数据库API  
数据库API

Python数据库API  
MySQL PostgreSQL ODBC  
数据库API  
数据库API

connect()数据库API  
数据库API  
数据库API

```
>>> import sqlite3
```

```
>>> db = sqlite3.connect('database.db')
>>>
```

cursor 객체를 생성하여 SQL 쿼리를 실행

```
>>> c = db.cursor()
>>> c.execute('create table portfolio (symbol text, shares
integer, price real)')
<sqlite3.Cursor object at 0x10067a730>
>>> db.commit()
>>>
```

데이터를 삽입하는 쿼리를 실행

```
>>> c.executemany('insert into portfolio values (?, ?, ?)',
stocks)
<sqlite3.Cursor object at 0x10067a730>
>>> db.commit()
>>>
```

데이터를 조회하는 쿼리를 실행

```
>>> for
row in
db.execute('select * from portfolio'):
...     print
(row)
...
```

```

('GOOG', 100, 490.1)
('AAPL', 50, 545.75)
('FB', 150, 7.45)
('HPQ', 75, 33.2)
>>>

```

How can we find all the rows in the portfolio table where the price is greater than or equal to a given value?

```

>>> min_price = 100
>>> for
    row in
db.execute('select * from portfolio where price >= ?',
            (min_price,)):
    ...     print
    (row)
    ...

('GOOG', 100, 490.1)
('AAPL', 50, 545.75)
>>>

```

## 6.8.3 Summary

In this section, we saw how to use the cursor object to execute SQL queries and fetch the results. We also saw how to use the cursor object to iterate over the results of a query.



Python  
datetime  
time  
system timestamps  
decimal  
Decimal

SQL  
Python  
%.format()  
SQL  
<http://xkcd.com/327>

?  
%s  
:0:1  
paramstyle

API  
object-relational  
mapper  
ORM  
SQLAlchemy  
<http://www.sqlalchemy.org>

Python과 SQL

## 6.9 문자열과 데이터베이스

### 6.9.1 문자열

문자열은 Python에서 가장 기본적인 데이터 타입 중 하나이다. 문자열은 단일 문자나 여러 문자를 포함할 수 있다.

### 6.9.2 문자열 인코딩

문자열을 바이트로 인코딩하는 방법은 여러 가지가 있다. `binascii` 모듈은 이 작업을 도와준다.

```
>>> # Initial byte string

>>> s = b'hello'
>>> # Encode as hex

>>> import binascii

>>> h = binascii.b2a_hex(s)
>>> h
b'68656c6c6f'
>>> # Decode back to bytes

>>> binascii.a2b_hex(h)
b'hello'
```

```
>>>
```

base64

```
>>> import base64
```

```
>>> h = base64.b16encode(s)
```

```
>>> h
```

```
b'68656C6C6F'
```

```
>>> base64.b16decode(h)
```

```
b'hello'
```

```
>>>
```

## 6.9.3

base64.b16decode()base64.b16encode()  
binascii

Unicode

```
>>> h = base64.b16encode(s)
```

```
>>> print
```

```
(h)
```

```
b'68656C6C6F'
```

```
>>> print
```

```
(h.decode('ascii'))
68656C6C6F
>>>
```

bytes.decode() a2b\_hex() bytes.decode() Unicode bytes.decode() ASCII

## 6.10 Base64

### 6.10.1

Base64

### 6.10.2

base64 b64encode() b64decode()

```
>>> # Some byte data

>>> s = b'hello'
>>> import base64

>>> # Encode as Base64
```

```
>>> a = base64.b64encode(s)
>>> a
b'aGVsbG8='
>>> # Decode from Base64

>>> base64.b64decode(a)
b'hello'
>>>
```

## 6.10.3 `base64`

`Base64`은 바이트 데이터를 64개의 가능한 문자 (대소문자, 숫자, `+`, `-`, `=`)로 인코딩하는 데 사용됩니다. `Base64` 인코딩된 데이터는 `Unicode` 문자열로 변환될 수 있습니다.

```
>>> a = base64.b64encode(s).decode('ascii')
>>> a
'aGVsbG8='
>>>
```

`Base64` 인코딩된 데이터는 `Unicode` 문자열로 변환될 수 있습니다. `Unicode` 문자열은 `ASCII` 문자열로 변환될 수 있습니다.

## 6.11 `base64`

### 6.11.1 `base64`

Python

## 6.11.2

Python

```
from struct import
Struct
def
write_records(records, format, f):

'''

Write a sequence of tuples to a binary file of structures.

'''

record_struct = Struct(format)
for
r in
```

```

records:

f.write(record_struct.pack(*r))

# Example

if
__name__ == '__main__':

records = [ (1, 2.3, 4.5),

(6, 7.8, 9.0),

(12, 13.4, 56.7) ]

with
open('data.b', 'wb') as
f:
    write_records(records, '<idd', f)

```

Python

```

from struct import
Struct
def
read_records(format, f):

```

```

        record_struct = Struct(format)
        chunks = iter(lambda
: f.read(record_struct.size), b'')
        return
(record_struct.unpack(chunk) for
chunk in
chunks)

# Example

if
__name__ == '__main__':
    with
open('data.b','rb') as
f:
    for
rec in
read_records('<idd', f):
    # Process rec

...

```

□□□□□□read()□□□□□□□□□□□□□□□□□□□□□□  
 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```

from struct import
Struct

```







```
(1, 2.0, 3.0)
>>>
```

pack() unpack()  
module-level functions

```
>>> import struct

>>> struct.pack('<idd', 1, 2.0, 3.0)
b'\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00@\x00\x00\x00\x00\x00\x00\x08@'
>>> struct.unpack('<idd', _)
(1, 2.0, 3.0)
>>>
```

Struct  
Struct

programming idioms  
read\_records()  
iter()  
5.8  
lambda:

```
f.read(record_struct.size)
b''
```

```
>>> f = open('data.b', 'rb')
>>> chunks = iter(lambda
: f.read(20), b'')
>>> chunks
<callable_iterator object at 0x10069e6d0>
>>> for
chk in
chunks:
...     print
(chk)
...

b'\x01\x00\x00\x00\xff\xff\xff\xff\x02@\x00\x00\x00\x00\x00\x00\x12@'
b'\x06\x00\x00\x00333333\x1f@\x00\x00\x00\x00\x00\x00"@'
b'\x0c\x00\x00\x00\xcd\xcc\xcc\xcc\xcc*\x9a\x99\x99\x99\x99YL@'
>>>
```

```

records

```

```
def
read_records(format, f):
    record_struct = Struct(format)
    while
True:
```





---

NumPy 是一个科学计算库，它提供了高性能的数组对象和大量的数学函数。NumPy 是 Python 中处理大规模数据集和科学计算的首选库。

```
>>> import numpy as np

>>> f = open('data.b', 'rb')
>>> records = np.fromfile(f, dtype='<i,<d,<d')
>>> records
array([(1, 2.3, 4.5), (6, 7.8, 9.0), (12, 13.4, 56.7)],
      dtype=[('f0', '<i4'), ('f1', '<f8'), ('f2', '<f8')])
>>> records[0]
(1, 2.3, 4.5)
>>> records[1]
(6, 7.8, 9.0)
>>>
```

NumPy 数组的形状（shape）属性返回一个元组，表示数组的维度。例如，一个 2D 数组的形状可能是 (10, 10)，表示它有 10 行和 10 列。NumPy 数组的形状属性是只读的，不能直接修改。要改变数组的形状，可以使用 `reshape()` 方法。

## 6.12 使用 NumPy 进行科学计算

### 6.12.1 数组的形状

NumPy 数组的形状（shape）属性返回一个元组，表示数组的维度。例如，一个 2D 数组的形状可能是 (10, 10)，表示它有 10 行和 10 列。NumPy 数组的形状属性是只读的，不能直接修改。要改变数组的形状，可以使用 `reshape()` 方法。

[zh.wikipedia.org/zh-cn/Shapefile](http://zh.wikipedia.org/zh-cn/Shapefile)

## 6.12.2 点数据

struct 数据类型在 Python 中通过 `collections.namedtuple` 实现。以下是一个示例，展示了如何定义一个名为 `Point` 的命名元组类，用于表示二维平面上的点。

```
polys = [
    [ (1.0, 2.5), (3.5, 4.0), (2.5, 1.5) ],
    [ (7.0, 1.2), (5.1, 3.0), (0.5, 7.5), (0.8, 9.0) ],
    [ (3.4, 6.3), (1.2, 0.5), (4.6, 9.2) ],
]
```

以下是一个示例，展示了如何定义一个名为 `Point` 的命名元组类，用于表示二维平面上的点。

偏移量	数据类型	变量名
0	int	0x1234
4	double	x
12	double	y
20	double	x
28	double	y





```
x, y in
flattened)

max_x = max(x for
x, y in
flattened)

min_y = min(y for
x, y in
flattened)

max_y = max(y for
x, y in
flattened)
with
open(filename, 'wb') as
f:

f.write(struct.pack('<iddddi',

0x1234,

min_x, min_y,

max_x, max_y,

len(polys)))
```

```

for
poly in
polys:

size = len(poly) * struct.calcsize('<dd')

f.write(struct.pack('<i', size+4))

for
pt in
poly:

f.write(struct.pack('<dd', *pt))
# Call it with our polygon data

write_polys('polys.bin', polys)

```

struct.unpack()
 unpack()
 pack()

```

import struct

def
read_polys(filename):

```

```
with
open(filename, 'rb') as
f:
    # Read the header

header = f.read(40)

file_code, min_x, min_y, max_x, max_y, num_polys = \
struct.unpack('<iddddi', header)

polys = []

for
n in
range(num_polys):

pbytes, = struct.unpack('<i', f.read(4))

poly = []

for
m in
range(pbytes // 16):

pt = struct.unpack('<dd', f.read(16))
```

```
poly.append(pt)

polys.append(poly)

return

polys
```

```

    read

```

```


```

```

    struct

```

```
import struct

class StructField
```

```
:  
    '''  
  
    Descriptor representing a simple structure field  
  
    '''  
  
def  
__init__(self, format, offset):  
  
    self.format = format  
  
    self.offset = offset  
  
def  
__get__(self, instance, cls):  
  
    if  
    instance is  
    None:  
  
    return  
    self  
  
    else  
    :
```

```

r = struct.unpack_from(self.format,
instance._buffer, self.offset)

return
r[0] if
len(r) == 1 else
r

class Structure
:

def
__init__(self, bytedata):

self._buffer = memoryview(bytedata)

```

descriptor  
 struct  
 format  
 offset  
 \_\_get\_\_()  
 struct.unpack\_from()  
 memoryview()

Structure  
 StructField  
 Structure  
 memoryview()

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
class PolyHeader  
(Structure):  
  
    file_code = StructField('<i', 0)  
  
    min_x = StructField('<d', 4)  
  
    min_y = StructField('<d', 12)  
  
    max_x = StructField('<d', 20)  
  
    max_y = StructField('<d', 28)  
  
    num_polys = StructField('<i', 36)
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> f = open('polys.bin', 'rb')  
>>> phead = PolyHeader(f.read(40))  
>>> phead.file_code == 0x1234  
True  
>>> phead.min_x  
0.5  
>>> phead.min_y  
0.5  
>>> phead.max_x  
7.0
```



```
>>> phead.max_y
9.2
>>> phead.num_polys
3
>>>
```

```
"""
    A class decorator that automatically creates StructField
    descriptors for each attribute of the class.
    StructField
    """
```

```
"""
    class decorator metaclass
    """
    """
    Structure
    """
```

```
class StructureMeta
    (type):
        """
        Metaclass that automatically creates StructField
        descriptors
        """

    def
    __init__(self, clsname, bases, clsdict):
```

```
fields = getattr(self, '_fields_', [])

byte_order = ''

offset = 0

for
format, fieldname in
fields:

if
format.startswith(('<', '>', '!', '@')):

byte_order = format[0]

format = format[1:]

format = byte_order + format

setattr(self, fieldname, StructField(format, offset))

offset += struct.calcsize(format)

setattr(self, 'struct_size', offset)
class Structure
(metaclass=StructureMeta):

def
```

```

__init__(self, bytedata):

self._buffer = bytedata


@classmethod

def
from_file(cls, f):

return
cls(f.read(cls.struct_size))

```

Structure

```

class PolyHeader
(Structure):

_fields_ = [

    ('<i', 'file_code'),

    ('d', 'min_x'),

    ('d', 'min_y'),

```

```
('d', 'max_x'),  
  
('d', 'max_y'),  
  
('i', 'num_polys')  
  
]
```

from\_file()  
PolyHeader.from\_file(f)  
file\_code == 0x1234  
min\_x  
min\_y  
max\_x  
max\_y  
num\_polys

```
>>> f = open('polys.bin', 'rb')  
>>> phead = PolyHeader.from_file(f)  
>>> phead.file_code == 0x1234  
True  
>>> phead.min_x  
0.5  
>>> phead.min_y  
0.5  
>>> phead.max_x  
7.0  
>>> phead.max_y  
9.2  
>>> phead.num_polys  
3  
>>>
```

PolyHeader  
file\_code  
min\_x  
min\_y  
max\_x  
max\_y  
num\_polys

```
class NestedStruct
:
    '''

    Descriptor representing a nested structure

    '''

def
__init__(self, name, struct_type, offset):

    self.name = name

    self.struct_type = struct_type

    self.offset = offset

def
__get__(self, instance, cls):

    if
instance is
None:

    return
self
```

```

else
:

data = instance._buffer[self.offset:

self.offset+self.struct_type.struct_size]

result = self.struct_type(data)
        # Save resulting structure back on instance to
avoid

        # further recomputation of this step

setattr(instance, self.name, result)

return
result
class StructureMeta
(type):
    '''

        Metaclass that automatically creates StructField
descriptors

        '''

def

```

```
__init__(self, clsname, bases, clsdict):

fields = getattr(self, '_fields_', [])

byte_order = ''

offset = 0

for
format, fieldname in
fields:

if
isinstance(format, StructureMeta):

setattr(self, fieldname,

NestedStruct(fieldname, format, offset))

offset += format.struct_size

else

:

if
format.startswith(('<', '>', '!', '@')):

byte_order = format[0]
```

```

format = format[1:]

format = byte_order + format

setattr(self, fieldname, StructField(format, offset))

offset += struct.calcsize(format)

setattr(self, 'struct_size', offset)

```

在 `NestedStruct` 中，我们使用 `memoryview` 来操作内存。
 [\[2\]](#) 我们使用 `memoryview` 来操作内存，
 因为 `memoryview` 可以让我们直接访问内存，
 而不需要知道内存的地址。
 在 `8.10` 中，我们使用 `memoryview` 来操作内存。

在 `8.10` 中，我们使用 `memoryview` 来操作内存。

```

class Point
(Structure):
    _fields_ = [
        ('<d', 'x'),

```





```
<__main__.Point object at 0x1006a48d0>
>>> phead.min.x
0.5
>>> phead.min.y
0.5
>>> phead.max.x
7.0
>>> phead.max.y
9.2
>>> phead.num_polys
3
>>>
```

6.11

```
class SizedRecord
:
def
__init__(self, bytedata):
self._buffer = memoryview(bytedata)
@classmethod
```

**def**

from\_file(cls, f, size\_fmt, includes\_size=True):

sz\_nbytes = struct.calcsize(size\_fmt)

sz\_bytes = f.read(sz\_nbytes)

sz, = struct.unpack(size\_fmt, sz\_bytes)

buf = f.read(sz - includes\_size \* sz\_nbytes)

**return**

cls(buf)

**def**

iter\_as(self, code):

**if**

isinstance(code, str):

s = struct.Struct(code)

**for**

off **in**

range(0, len(self.\_buffer), s.size):

**yield**

```

s.unpack_from(self._buffer, off)

elif
instance(code, StructureMeta):

size = code.struct_size

for
off in
range(0, len(self._buffer), size):

data = self._buffer[off:off+size]

yield
code(data)

```

SizedRecord.from\_file()
 includes\_size

```

>>> f = open('polys.bin', 'rb')
>>> phead = PolyHeader.from_file(f)
>>> phead.num_polys
3
>>> polydata = [ SizedRecord.from_file(f, '<i')

```

```

...             for
n in
range(phead.num_polys) ]
>>> polydata
[<__main__.SizedRecord object at 0x1006a4d50>,
 <__main__.SizedRecord object at 0x1006a4f50>,
 <__main__.SizedRecord object at 0x10070da90>]
>>>

```

SizedRecord  
 iter\_as()  
 Structure

```

>>> for
n, poly in
enumerate(polydata):
...     print
('Polygon', n)
...     for
p in
poly.iter_as('<dd'):
...         print
(p)
...

Polygon 0
(1.0, 2.5)
(3.5, 4.0)

```

```
(2.5, 1.5)
Polygon 1
(7.0, 1.2)
(5.1, 3.0)
(0.5, 7.5)
(0.8, 9.0)
Polygon 2
(3.4, 6.3)
(1.2, 0.5)
(4.6, 9.2)
>>>
```

```
>>> for
n, poly in
enumerate(polydata):
...     print
('Polygon', n)
...     for
p in
poly.iter_as(Point):
...         print
(p.x, p.y)
...
```

```
Polygon 0
1.0 2.5
3.5 4.0
2.5 1.5
Polygon 1
7.0 1.2
5.1 3.0
0.5 7.5
0.8 9.0
Polygon 2
3.4 6.3
1.2 0.5
4.6 9.2
```

```
>>>
```

```
def read_polys():  
    pass
```

```
class Point
```

```
(Structure):
```

```
    _fields_ = [  
        ('<d', 'x'),  
        ('d', 'y')  
    ]
```

```
class PolyHeader
```

```
(Structure):
```

```
    _fields_ = [  
        ('<i', 'file_code'),  
        (Point, 'min'),  
        (Point, 'max'),  
        ('i', 'num_polys')  
    ]
```

```
def
```

```
read_polys(filename):
```

```
    polys = []  
    with
```

```
open(filename, 'rb') as
```

```
f:
```

```
    phead = PolyHeader.from_file(f)  
    for
```

```
n in
```

```
range(phead.num_polys):
```

```
    rec = SizedRecord.from_file(f, '<i')  
    poly = [ (p.x, p.y)  
              for
```

```

p in
rec.iter_as(Point) ]
    polys.append(poly)
    return
polys

```

## 6.12.3 内存视图

内存视图（memoryview）是 Python 3 中引入的一个新类型，它提供了一种高效且安全的方式来访问内存数据。内存视图对象可以看作是对内存中一段连续数据的引用，它不拥有数据，只是指向数据。因此，内存视图的创建和销毁都不会导致内存的分配和释放，这使得它在处理大量数据时非常高效。

内存视图的创建非常简单，只需要使用 `memoryview()` 函数即可。该函数接受一个可内存映射的对象（如文件、字节数组、缓冲区等）作为参数，并返回一个内存视图对象。内存视图对象支持切片操作，可以方便地访问内存中的特定部分。此外，内存视图还支持一些高级操作，如设置步长、设置起始位置等，这使得它在处理复杂内存布局时非常灵活。

内存视图的一个重要特性是它可以与 C 语言的结构体（struct）进行交互。通过 `StructField` 和 `StructureMeta` 类，我们可以方便地定义和管理结构体的成员。内存视图对象可以通过 `_fields_` 属性访问结构体的成员，这使得在 C 扩展模块中使用 Python 内存视图变得非常简单。此外，内存视图还支持与 NumPy 数组的交互，这使得在科学计算和数据分析领域的应用非常广泛。



A grid consisting of two horizontal rows of empty square boxes. The top row contains 20 boxes, and the bottom row also contains 20 boxes, aligned directly below the first row.

StructureMeta  
<  
>

```
class ShapeFile
```

```
(Structure):
    _fields_ = [ ('>i', 'file_code'), # Big endian [padding]
                  ('20s', 'unused'),
                  ('i', 'file_length'),
                  ('<i', 'version'), # Little endian [padding]

                  ('i', 'shape_type'),
                  ('d', 'min_x'),
                  ('d', 'min_y'),
                  ('d', 'max_x'),
                  ('d', 'max_y'),
                  ('d', 'min_z'),
                  ('d', 'max_z'),
                  ('d', 'min_m'),
                  ('d', 'max_m') ]
```

memoryview() 返回一个 memoryview 对象。memoryview 对象可以访问内存缓冲区，并且可以切片。memoryview 对象可以访问内存缓冲区，并且可以切片。memoryview 对象可以访问内存缓冲区，并且可以切片。memoryview 对象可以访问内存缓冲区，并且可以切片。memoryview 对象可以访问内存缓冲区，并且可以切片。

8.13 8.10 NestedStruct 9.19 StructureMeta Python ctypes

## 6.13

### 6.13.1

### 6.13.2

`Pandas` <http://pandas.pydata.org>

Pandas  
<https://data.cityofchicago.org/Service-Requests/311-Service-Requests-Rodent-Baiting/97t6-zrhs> CSV  
 74 000

```
>>> import pandas

>>> # Read a CSV file, skipping last line

>>> rats = pandas.read_csv('rats.csv', skip_footer=1)
>>> rats
<class 'pandas.core.frame.DataFrame'>
Int64Index: 74055 entries, 0 to 74054
Data columns:
Creation Date                74055 non-null values
Status                      74055 non-null values
Completion Date              72154 non-null values
Service Request Number      74055 non-null values
Type of Service Request     74055 non-null values
Number of Premises Baited    65804 non-null values
Number of Premises with Garbage 65600 non-null values
Number of Premises with Rats 65752 non-null values
Current Activity            66041 non-null values
Most Recent Action          66023 non-null values
Street Address              74055 non-null values
ZIP Code                    73584 non-null values
X Coordinate                74043 non-null values
Y Coordinate                74043 non-null values
Ward                       74044 non-null values
Police District             74044 non-null values
Community Area              74044 non-null values
```

```

Latitude          74043 non-null values
Longitude         74043 non-null values
Location          74043 non-null values
dtypes: float64(11), object(9)
>>> # Investigate range of values for a certain field

>>> rats['Current Activity'].unique()
array([nan, Dispatch Crew, Request Sanitation Inspector],
      dtype=object)
>>> # Filter the data

>>> crew_dispatched = rats[rats['Current Activity'] ==
'Dispatch Crew']
>>> len(crew_dispatched)
65676
>>>
>>> # Find 10 most rat-infested ZIP codes in Chicago

>>> crew_dispatched['ZIP Code'].value_counts()[:10]
60647      3837
60618      3530
60614      3284
60629      3251
60636      2801
60657      2465
60641      2238
60609      2206
60651      2152
60632      2071
>>>
>>> # Group by completion date

>>> dates = crew_dispatched.groupby('Completion Date')
<pandas.core.groupby.DataFrameGroupBy object at 0x10d0a2a10>
>>> len(dates)
472
>>>
>>> # Determine counts on each day

>>> date_counts = dates.size()

```

>>>

Pandas

□□□□□□□□Pandas□□□□□□□□

□Wes McKinney□□□Python for Data  
Analysis□O'Reilly□□□□□□□□□□□□□□□□

---

[1] Num-Premises□□-□□□□Python□□□□□  
□□——□□□

[2] □□C++□□placement new□□□——□□□

## 7 列表

```
def avg(*args):
    """Return the average of the arguments"""
    return sum(args) / len(args)

# Sample use
```

## 7.1 列表操作

### 7.1.1 列表

```
list = []
```

### 7.1.2 列表操作

```
list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] * 2
list = list + [11, 12, 13, 14, 15]
```

```
def
avg(first, *rest):
    return
    (first + sum(rest)) / (1 + len(rest))
# Sample use
```

```
avg(1, 2)                # 1.5
```

```
avg(1, 2, 3, 4)          # 2.5
```

```
def rest(*args):
    """Return a list of the arguments, excluding the first one.
    If no arguments are given, return an empty list.
    """
```

```
    return list(args[1:])

```

```
import html
```

```
def
```

```
make_element(name, value, **attrs):
    keyvals = [' %s="%s"' % item for
```

```
item in
```

```
attrs.items()]
    attr_str = ''.join(keyvals)
    element = '<{name}{attrs}>{value}</{name}>'.format(
        name=name,
        attrs=attr_str,
        value=html.escape(value))
```

```
    return
```

```
element
```

```
# Example
```



```
# Creates '<item size="large" quantity="6">Albatross</item>'
make_element('item', 'Albatross', size='large', quantity=6)

# Creates '<p><spam></p>'

make_element('p', '<spam>')
```

```
    attrs = {}
    for key, value in kwargs.items():
        if key in attrs:
            attrs[key] = value
        else:
            attrs[key] = value
```

```
    for key, value in kwargs.items():
        if key in attrs:
            attrs[key] = value
        else:
            attrs[key] = value
```

```
def
anyargs(*args, **kwargs):
    print
    (args)          # A tuple

    print
    (kwargs)        # A dict
```

```
    args = {}
    kwargs = {}
```

## 7.1.3 位置

位置引数 \* 変数名 \*\* 辞書型引数

位置引数 \* 変数名 \*\* 辞書型引数

位置引数 \* 変数名 \*\* 辞書型引数

```
def
```

```
a(x, *args, y):  
    pass
```

```
def
```

```
b(x, *args, y, **kwargs):  
    pass
```

keyword-only 変数名 \*args

7.2

## 7.2 位置

### 7.2.1 位置

位置引数 \* 変数名 \*\* 辞書型引数

## 7.2.2 関数

関数呼び出しの構文は、`関数名(引数1, 引数2, ...)` のように記述する。引数は、関数の定義で指定したパラメータに値を代入する。関数は、`return` 文で値を返すことができる。

```
def
recv(maxsize, *, block):
    'Receives a message'
    pass

recv(1024, True)          # TypeError

recv(1024, block=True)    # Ok
```

関数の定義は、`def 関数名(パラメータ1, パラメータ2, ...):` のように記述する。関数の体は、`:` の直下で記述する。

```
def
minimum(*values, clip=None):
    m = min(values)
    if
clip is not
None:
        m = clip if
clip > m else
```

```

m
    return
m
minimum(1, 5, 2, -5, 10)           # Returns -5

minimum(1, 5, 2, -5, 10, clip=0)   # Returns 0

```

## 7.2.3 参数

函数 `recv()` 支持 `keyword-only` 参数，即只通过关键字传递的参数。

```
msg = recv(1024, False)
```

函数 `recv()` 的第二个参数 `False` 表示非阻塞模式，即如果数据不可用，函数会立即返回，而不是等待数据可用。

```
msg = recv(1024, block=False)
```

函数 `recv()` 还支持 `**kwargs` 参数，即通过关键字传递的参数。

```
>>> help(recv)
Help on function recv in module __main__:
recv(maxsize, *, block)
    Receives a message
```

keyword-only arguments

arguments after \*args

\*\*kwargs arguments 9.11

## 7.3

### 7.3.1

arguments

### 7.3.2

arguments

```
def
add(x:int, y:int) -> int:
    return
    x + y
```



9.20

## 7.4

### 7.4.1

### 7.4.2

```
>>> def
myfun():
...     return
1, 2, 3
...
>>> a, b, c = myfun()
>>> a
1
>>> b
2
>>> c
3
```

### 7.4.3

□□□□myFun()□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□

```
>>> a = (1, 2)           # With parentheses

>>> a
(1, 2)
>>> b = 1, 2           # Without parentheses

>>> b
(1, 2)
>>>
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□1.1□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□

```
>>> x = myfun()
>>> x
(1, 2, 3)
>>>
```

□□x□□□□□□□□

## 7.5 □□□□□□□□□□

### 7.5.1 □□



~~~~~  
~~~~~

## 7.5.2 ~~~~

~~~~~——~~~~~  
~~~~~

```
def
spam(a, b=42):
    print
(a, b)
spam(1)          # Ok. a=1, b=42
spam(1, 2)       # Ok. a=1, b=2
```

~~~~~  
None~~~~~

```
# Using a list as a default value

def
spam(a, b=None):
    if
```

■ ■ ■

```
>>> spam(1)
No b value supplied
>>> spam(1, 2)           # b = 2

>>> spam(1, None)       # b = None

>>>
```

None

## 7.5.3

```
>>> x = 42
>>> def
spam(a, b=x):
...     print
(a, b)
...

>>> spam(1)
1 42
>>> x = 23      # Has no effect

>>> spam(1)
1 42
>>>
```

None
True
False

```
def
spam(a, b=[]):           # NO!

    ...
```

```
>>> def
spam(a, b=[]):
...
    print
(b)
...
    return
b
...

>>> x = spam(1)
>>> x
[]
>>> x.append(99)
>>> x.append('Yow!')
>>> x
[99, 'Yow!']
>>> spam(1)                # Modified list gets returned!
```

```
[99, 'Yow! ']  
>>>
```

[illegible]

```

None is

```

```
def
spam(a, b=None):
    if not
b:          # NO! Use 'b is None' instead
            b = []
    ...
```

```

None
False
0
False

```

```
>>> spam(1)                # OK

>>> x = []
>>> spam(1, x)              # Silent error. x value overwritten by
                             default
```

```
>>> spam(1, 0)          # Silent error. 0 ignored

>>> spam(1, '')         # Silent error. '' ignored

>>>
```

—————

None

0

False

object()

\_no\_value

\_no\_value

\_no\_value

object()

Python

object

\_\_dict\_\_

## 7.6 匿名函数

### 7.6.1 列表

对列表元素排序时，使用 `sort()` 方法，可以指定一个函数，`def` 关键字，函数名，参数列表，函数体，函数名后跟冒号，函数体最后一行以 `return` 语句结束。

### 7.6.2 匿名函数

匿名函数使用 `lambda` 关键字，函数名，参数列表，函数体，函数名后跟冒号，函数体最后一行以 `return` 语句结束。

```
>>> add = lambda  
x, y: x + y  
>>> add(2,3)  
5  
>>> add('hello', 'world')  
'helloworld'  
>>>
```

匿名函数 `lambda` 函数名，参数列表，函数体，函数名后跟冒号，函数体最后一行以 `return` 语句结束。

```
>>> def  
add(x, y):  
...  
    return
```

```
x + y
...

>>> add(2,3)
5
>>>
```

lambda 表达式是 Python 语言中一种小的匿名函数，它可以在一行内完成定义和调用，而不需要事先定义一个完整的函数。

```
>>> names = ['David Beazley', 'Brian Jones',
...          'Raymond Hettinger', 'Ned Batchelder']
>>> sorted(names, key=lambda
name: name.split()[-1].lower())
['Ned Batchelder', 'David Beazley', 'Raymond Hettinger',
'Brian Jones']
>>>
```

## 7.6.3 lambda 表达式

lambda 表达式是 Python 语言中一种小的匿名函数，它可以在一行内完成定义和调用，而不需要事先定义一个完整的函数。lambda 表达式通常用于需要快速定义一个简单函数的场合，例如在调用某些函数时作为参数使用。

lambda 表达式在 Python 中非常有用，尤其是在需要快速定义一个简单函数的场合。lambda 表达式通常用于调用某些函数时作为参数使用。



lambda 表达式是 Python 中一种特殊的数据类型，它提供了一种简洁的方式来定义函数。lambda 表达式通常用于需要快速定义简单函数的场合，例如在列表推导式、字典推导式以及函数参数中。

## 7.7 lambda 表达式

### 7.7.1 定义

lambda 表达式的基本语法如下：  
lambda 参数: 表达式

### 7.7.2 使用

以下是一个简单的例子：

```
>>> x = 10
>>> a = lambda y: x + y

>>> x = 20
>>> b = lambda y: x + y

>>> a(10)
30
>>> b(10)
30
```

在这个例子中，我们定义了两个 lambda 函数 a 和 b。函数 a 的定义是 lambda y: x + y，其中 x 的值为 10。函数 b 的定义是 lambda y: x + y，其中 x 的值为 20。当我们调用 a(10) 时，返回的结果是 30；当我们调用 b(10) 时，返回的结果也是 30。

```
>>> a(10)
30
```

```
>>> b(10)
30
>>>
```

lambda 函数可以接受任何类型的参数，并且可以接受任何类型的返回值。lambda 函数可以接受任何类型的参数，并且可以接受任何类型的返回值。

```
>>> x = 15
>>> a(10)
25
>>> x = 3
>>> a(10)
13
>>>
```

lambda 函数可以接受任何类型的参数，并且可以接受任何类型的返回值。lambda 函数可以接受任何类型的参数，并且可以接受任何类型的返回值。

```
>>> x = 10
>>> a = lambda
y, x=x: x + y
>>> x = 20
>>> b = lambda
y, x=x: x + y
>>> a(10)
20
>>> b(10)
30
>>>
```

## 7.7.3 `lambda`

`lambda` 表达式是 Python 中一种特殊表达式，它允许你在一个表达式中定义一个函数。它的语法是：  
`lambda [parameters]: expression`  
其中 `[parameters]` 是函数参数列表，`expression` 是函数体表达式。它返回一个函数对象，可以在需要时调用。

```
>>> funcs = [lambda  
x: x+n for  
n in  
range(5)]  
>>> for  
f in  
funcs:  
...  
    print  
(f(0))  
...  
  
4  
4  
4  
4  
4  
>>>
```

上面代码中，`funcs` 是一个列表，包含 5 个 `lambda` 表达式。每个表达式都接受一个参数 `x`，并返回 `x+n`，其中 `n` 是 0 到 4 的整数。因此，`funcs` 列表中的每个元素都是一个函数对象，调用时返回 4。

```
>>> funcs = [lambda
x, n=n: x+n for
n in
range(5)]
>>> for
f in
funcs:
...
    print
(f(0))
...
0
1
2
3
4
>>>
```

n

## 7.8 □□□N□□□□□□□□□□□□□□

□□□

### 7.8.1 练习

Python  
Python

## 7.8.2

`functools.partial()`  
`functools.partial()`  
`functools.partial()`  
`functools.partial()`  
`functools.partial()`

```
def  
spam(a, b, c, d):  
    print  
(a, b, c, d)
```

`functools.partial()`

```
>>> from functools import  
partial  
>>> s1 = partial(spam, 1)          # a = 1  
  
>>> s1(2, 3, 4)  
1 2 3 4  
>>> s1(4, 5, 6)  
1 4 5 6  
>>> s2 = partial(spam, d=42)      # d = 42  
  
>>> s2(1, 2, 3)
```

```

1 2 3 42
>>> s2(4, 5, 5)
4 5 5 42
>>> s3 = partial(spam, 1, 2, d=42)      # a = 1, b = 2, d = 42

>>> s3(3)
1 2 3 42
>>> s3(4)
1 2 4 42
>>> s3(5)
1 2 5 42
>>>

```

partial()은 함수의 인자 중 일부를 고정하고, 나머지를 변수로 받아들이는 함수이다.

partial()은 함수의 인자 중 일부를 고정하고, 나머지를 변수로 받아들이는 함수이다.

## 7.8.3 람다

람다는 함수를 정의하는 한 가지 방법이다.

람다는 함수를 정의하는 한 가지 방법이다.

```

points = [ (1, 2), (3, 4), (5, 6), (7, 8) ]

import math

def

```

```
distance(p1, p2):
    x1, y1 = p1
    x2, y2 = p2
    return
math.hypot(x2 - x1, y2 - y1)
```

sort() 的 key 参数可以指定一个函数，该函数接收一个元素并返回一个值，该值将用于排序。在这个例子中，我们使用 partial() 函数来创建一个新的函数，该函数接收一个点并返回其距离。然后，我们将这个新函数作为 key 参数传递给 sort() 函数。

```
>>> pt = (4, 3)
>>> points.sort(key=partial(distance,pt))
>>> points
[(3, 4), (1, 2), (5, 6), (7, 8)]
>>>
```

partial() 函数是 functools 模块中的一个函数，它用于创建一个新的函数，该函数接收一个点并返回其距离。然后，我们将这个新函数作为 key 参数传递给 sort() 函数。

```
def
output_result(result, log=None):
    if
log is not
None:
    log.debug('Got: %r', result)
```

```

# A sample function

def
add(x, y):
    return
x + y

if
__name__ == '__main__':
    import logging

    from multiprocessing import
Pool
    from functools import
partial

    logging.basicConfig(level=logging.DEBUG)
    log = logging.getLogger('test')

    p = Pool()
    p.apply_async(add, (3, 4), callback=partial(output_result,
log=log))
    p.close()
    p.join()

```

apply\_async()
 partial()
 multiprocessing
 —

socketserver



## Simple echo server

```
from socketserver import
StreamRequestHandler, TCPServer

class EchoHandler
(StreamRequestHandler):
    def
handle(self):
    for
line in
self.rfile:
    self.wfile.write(b'GOT:' + line)

serv = TCPServer(('', 15000), EchoHandler)
serv.serve_forever()
```

## Simple EchoHandler

### \_\_init\_\_()

```
class EchoHandler
(StreamRequestHandler):
    # ack is added keyword-only argument. *args, **kwargs are

    # any normal parameters supplied (which are passed on)

    def
__init__(self, *args, ack, **kwargs):
```

```

        self.ack = ack
        super().__init__(*args, **kwargs)
    def
handle(self):
    for
line in
self.rfile:
    self.wfile.write(self.ack + line)

```

TCPServer

```

Exception happened during processing of request from
('127.0.0.1', 59834)
Traceback (most recent call last):
...
TypeError: __init__() missing 1 required keyword-only
argument: 'ack'

```

socketserver
 partial()
 partial()
 ack

```

from functools import
partial
serv = TCPServer('', 15000), partial(EchoHandler,
ack=b'RECEIVED:')
serv.serve_forever()

```

\_\_init\_\_()ack  
keyword- only  
keyword-only7.2

lambda partial()  
points.sort(key=

```
points.sort(key=
p: distance(pt, p))
p.apply_async(add, (3, 4), callback=
result: output_result(result,log))
serv = TCPServer(('', 15000),
                 lambda
*args, **kwargs: EchoHandler(*args,
ack=b'RECEIVED:',
**kwargs))
```

partial()  
partial()

## 7.9

### 7.9.1

\_\_init\_\_() 方法返回一个  
字典，字典的键是模板中的变量名，值是变量的值。

## 7.9.2 模板

模板是一个字符串，其中包含一些变量名，这些变量名在模板被渲染时会被替换成变量的值。模板的渲染是通过 `closure` 方法实现的，该方法返回一个 `URL` 对象。

```
from urllib.request import
urlopen

class UrlTemplate
:
    def
__init__(self, template):
    self.template = template
    def
open(self, **kwargs):
        return
urlopen(self.template.format_map(kwargs))

# Example use. Download stock data from yahoo

yahoo = UrlTemplate('http://finance.yahoo.com/d/quotes.csv?s=
{names}&f={fields}')
for
line in
yahoo.open(names='IBM,AAPL,FB', fields='sl1clv'):
```

```
    print
(line.decode('utf-8'))
```

□□□□□□□□□□□□□□□□

```
def
urltemplate(template):
    def
opener(**kwargs):
    return
urlopen(template.format_map(kwargs))
    return
opener
# Example use

yahoo = urltemplate('http://finance.yahoo.com/d/quotes.csv?s=
{names}&f={fields}')
for
line in
yahoo(names='IBM,AAPL,FB', fields='sl1c1v'):
    print
(line.decode('utf-8'))
```

## 7.9.3 □□

```
template<...> void open() { ... }
```

[illegible][illegible]

## 7.10

### 7.10.1 ☐☐

The diagram consists of two horizontal rows of squares. The top row contains 20 squares, and the bottom row also contains 20 squares. The top row is shifted to the right by one position relative to the bottom row. This visualizes a shift operation where the elements of the top array are moved one position to the right.

## 7.10.2 ☐☐☐☐

[illegible]

```
def
apply_async(func, args, *, callback):
    # Compute the result

    result = func(*args)

    # Invoke the callback with the result

    callback(result)
```

```
>>> def
print_result(result):
...
    print
('Got:', result)
...

>>> def
add(x, y):
...
    return
x + y
```

```
...

>>> apply_async(add, (2, 3), callback=print_result)
Got: 5
>>> apply_async(add, ('hello', 'world'),
callback=print_result)
Got: helloworld
>>>
```

print\_result() 接收一个结果参数 result，并打印它。

bound-method 绑定方法，它接收一个结果参数，并打印它。

```
class ResultHandler
:
    def
__init__(self):
    self.sequence = 0
    def
handler(self, result):
    self.sequence += 1
    print
('[{}}] Got: {}'.format(self.sequence, result))
```



handler

```
>>> r = ResultHandler()
>>> apply_async(add, (2, 3), callback=r.handler)
[1] Got: 5
>>> apply_async(add, ('hello', 'world'), callback=r.handler)
[2] Got: helloworld
>>>
```

```
def
make_handler():
    sequence = 0
    def
handler(result):
    nonlocal sequence
    sequence += 1
    print
('[{}}] Got: {}'.format(sequence, result))
return
handler
```

```
>>> handler = make_handler()
>>> apply_async(add, (2, 3), callback=handler)
[1] Got: 5
>>> apply_async(add, ('hello', 'world'), callback=handler)
[2] Got: helloworld
```

```
>>>
```

Coroutine routines  
are implemented as

```
def
make_handler():
    sequence = 0
    while
True:
        result = yield

        sequence += 1
        print

('[{}}] Got: {}'.format(sequence, result))
```

Coroutine routines  
are implemented as

```
>>> handler = make_handler()
>>> next(handler)           # Advance to the yield

>>> apply_async(add, (2, 3), callback=handler.send)
[1] Got: 5
>>> apply_async(add, ('hello', 'world'),
callback=handler.send)
[2] Got: helloworld
>>>
```

partial() 7.8

```
>>> class SequenceNo
:
...     def
__init__(self):
...         self.sequence = 0
...
>>> def
handler(result, seq):
...     seq.sequence += 1
...     print
('[{}}] Got: {}'.format(seq.sequence, result))
...
>>> seq = SequenceNo()
>>> from functools import
partial
>>> apply_async(add, (2, 3), callback=partial(handler,
seq=seq))
[1] Got: 5
>>> apply_async(add, ('hello', 'world'),
callback=partial(handler, seq=seq))
[2] Got: helloworld
>>>
```

## 7.10.3

非同期に呼び出す関数に、`callback`という引数を渡す。これは、関数の実行が完了したときに呼び出される関数で、`callback`は、`lambda`で定義する。

例えば、`add`関数を非同期に呼び出す場合、`callback`として、`lambda x, y: print(x + y)`を渡す。すると、`add`関数の実行が完了したときに、`print(x + y)`が実行される。これは、`lambda`で定義した関数を、`callback`として渡している。

また、`nonlocal`というキーワードも、`sequence`という変数に対して、`nonlocal`で宣言する必要がある。これは、`sequence`という変数が、`nonlocal`で宣言されていることを示す。

例えば、`nonlocal`というキーワードを使って、`sequence`という変数を宣言する。これは、`sequence`という変数が、`nonlocal`で宣言されていることを示す。また、`Python`という変数も、`nonlocal`で宣言する必要がある。これは、`Python`という変数が、`nonlocal`で宣言されていることを示す。また、`next()`という関数も、`nonlocal`で宣言する必要がある。これは、`next()`という関数が、`nonlocal`で宣言されていることを示す。

また、`partial()`という関数も、`lambda`という関数で定義する必要がある。これは、`partial()`という関数が、`lambda`で定義されていることを示す。

```
>>> apply_async(add, (2, 3), callback=lambda
```

```
r: handler(r, seq))
[1] Got: 5
>>>
```

partial() 7.8

## 7.11

### 7.11.1

7.10

### 7.11.2

7.10

```
def
apply_async(func, args, *, callback):
    # Compute the result

    result = func(*args)

    # Invoke the callback with the result
```

```
callback(result)
```

```
class Async
    inline_async
```

```
from queue import
Queue
from functools import
wraps

class Async
:
    def

__init__(self, func, args):
    self.func = func
    self.args = args

def
inline_async(func):
    @wraps(func)
    def
wrapper(*args):
        f = func(*args)
        result_queue = Queue()
        result_queue.put(None)
        while
True:
            result = result_queue.get()
            try
:

```

```

        a = f.send(result)
        apply_async(a.func, a.args,
callback=result_queue.put)
    except StopIteration

:
        break

    return
wrapper

```

yield

```

def
add(x, y):
    return
x + y
@inlined_async
def
test():
    r = yield
Async(add, (2, 3))
    print
(r)
    r = yield
Async(add, ('hello', 'world'))
    print
(r)
    for

```

```

n in
range(10):
    r = yield
Async(add, (n, n))
    print
(r)
    print
('Goodbye')

```

□□□□test()□□□□□□□□□□□□

```

5
helloworld
0
2
4
6
8
10
12
14
16
18
Goodbye

```

□□□□□□□□□□yield□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□

## 7.11.3 □□



1. 在 `__init__` 方法中，我们首先创建了一个 `Queue` 对象，用于存储任务。  
 2. 然后，我们创建了一个 `ThreadPoolExecutor` 对象，用于执行任务。

3. 在 `__call__` 方法中，我们使用 `apply_async` 方法将任务添加到任务队列中。  
 4. 最后，我们使用 `get` 方法从任务队列中获取任务的结果。  
 5. 在 `__del__` 方法中，我们使用 `shutdown` 方法关闭线程池。

6. 在 `__call__` 方法中，我们使用 `yield` 语句来返回任务的结果。  
 7. 在 `__del__` 方法中，我们使用 `__next__` 方法来返回任务的结果。  
 8. 在 `__del__` 方法中，我们使用 `send` 方法来返回任务的结果。

9. 在 `__call__` 方法中，我们使用 `inline_async` 方法来返回任务的结果。  
 10. 在 `__call__` 方法中，我们使用 `yield` 语句来返回任务的结果。  
 11. 在 `__call__` 方法中，我们使用 `None` 来返回任务的结果。  
 12. 在 `__call__` 方法中，我们使用 `yield` 语句来返回任务的结果。  
 13. 在 `__call__` 方法中，我们使用 `Async` 来返回任务的结果。  
 14. 在 `__call__` 方法中，我们使用 `apply_async` 方法来返回任务的结果。  
 15. 在 `__call__` 方法中，我们使用 `put` 方法来返回任务的结果。

16. 在 `__call__` 方法中，我们使用 `get` 方法来返回任务的结果。  
 17. 在 `__call__` 方法中，我们使用 `put` 方法来返回任务的结果。  
 18. 在 `__call__` 方法中，我们使用 `apply_async` 方法来返回任务的结果。  
 19. 在 `__call__` 方法中，我们使用 `apply_async` 方法来返回任务的结果。

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
if
__name__ == '__main__':
    import multiprocessing

    pool = multiprocessing.Pool()
    apply_async = pool.apply_async

    # Run the test function

    test()
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□contextlib□□□□  
@contextmanager□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□yield□□□□□□□□□□□□  
Twisted□□<http://twistedmatrix.com> □□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□

# 7.12 □□□□□□□□□□□□□□□□

## 7.12.1 ☐ ☐

## 7.12.2 ☐☐☐☐

```

    accessor function
getter/setter

```

```
def
sample():
    n = 0
    # Closure function

    def
    func():
        print
        ('n=', n)

        # Accessor methods for n

    def
    get_n():
        return
n
```



□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
import sys

class ClosureInstance
:
    def
__init__(self, locals=None):
    if
locals is
None:
        locals = sys._getframe(1).f_locals
        # Update instance dictionary with callables

        self.__dict__.update((key,value) for
key, value in
locals.items()
                                if
callable(value) )
        # Redirect special methods

    def
__len__(self):
        return
self.__dict__['__len__']()
# Example use
```

```

def
Stack():
    items = []

    def
push(item):
    items.append(item)

    def
pop():
    return
items.pop()

    def
__len__():
    return
len(items)
    return
ClosureInstance()

```

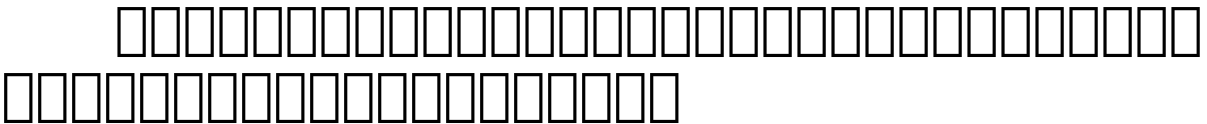
□□□□□□□□□□□□□□□□□□□□

```

>>> s = Stack()
>>> s
<__main__.ClosureInstance object at 0x10069ed10>
>>> s.push(10)
>>> s.push(20)
>>> s.push('Hello')
>>> len(s)
3
>>> s.pop()
'Hello'
>>> s.pop()

```

```
20
>>> s.pop()
10
>>>
```



```
class Stack2
:
    def
__init__(self):
    self.items = []

    def
push(self, item):
    self.items.append(item)

    def
pop(self):
    return
self.items.pop()

    def
__len__(self):
    return
len(self.items)
```



```

>>> from timeit import
timeit
>>> # Test involving closures

>>> s = Stack()
>>> timeit('s.push(1);s.pop()', 'from __main__ import s')
0.9874754269840196
>>> # Test involving a class

>>> s = Stack2()
>>> timeit('s.push(1);s.pop()', 'from __main__ import s')
1.0707052160287276
>>>

```

8%
   
 self

Raymond Herring
   
 “”
   
 ClosureInstance
   
 \_\_len\_\_()

8%
   
 self



[illegible]

## 第8章 自定义类型

在本章中，我们将学习如何定义自己的数据类型。Python 提供了丰富的内置数据类型，但有时我们需要创建自己的数据类型来满足特定的需求。本章将介绍如何定义类、如何创建自定义数据类型以及如何使用它们。

### 8.1 自定义数据类型

#### 8.1.1 类

在 Python 中，类（Class）是创建对象的蓝图。类定义了对象的属性和方法。我们可以使用类来创建多个对象，每个对象都是类的实例。

#### 8.1.2 类方法

类方法（Class Method）是定义在类中的方法，它们可以访问类的属性。类方法通常用于操作类的状态。在 Python 中，类方法使用 `@classmethod` 装饰器来定义。类方法的第一个参数是类本身，而不是实例。

```
class Pair:
    def
__init__(self, x, y):
    self.x = x
    self.y = y
    def
__repr__(self):
```

```

        return

    'Pair({0.x!r}, {0.y!r})'.format(self)
    def
__str__(self):
    return
'({0.x!s}, {0.y!s})'.format(self)

```

我们实现 `__repr__()` 来生成对象的代码 representation  
 我们实现 `__str__()` 来生成对象的字符串 representation  
 我们实现 `__repr__()` 来生成对象的代码 representation  
 我们实现 `__str__()` 来生成对象的字符串 representation  
 我们实现 `__repr__()` 来生成对象的代码 representation  
 我们实现 `__str__()` 来生成对象的字符串 representation

```

>>> p = Pair(3, 4)
>>> p
Pair(3, 4)          # __repr__() output
>>> print
(p)
(3, 4)              # __str__() output
>>>

```

我们实现 `__repr__()` 来生成对象的代码 representation  
 我们实现 `__str__()` 来生成对象的字符串 representation  
 我们实现 `__repr__()` 来生成对象的代码 representation  
 我们实现 `__str__()` 来生成对象的字符串 representation  
 我们实现 `__repr__()` 来生成对象的代码 representation  
 我们实现 `__str__()` 来生成对象的字符串 representation

```

>>> p = Pair(3, 4)
>>> print
('p is {0!r}'.format(p))
p is Pair(3, 4)

```

```
>>> print
('p is {0}'.format(p))
p is (3, 4)
>>>
```

### 8.1.3 字符串

\_\_repr\_\_() 和 \_\_str\_\_() 方法返回字符串表示。\_\_repr\_\_() 返回的字符串应该能够重新创建对象，而 \_\_str\_\_() 返回的字符串应该是一个可读的字符串表示。

\_\_repr\_\_() 方法返回的字符串应该能够重新创建对象，即 eval(repr(x)) == x。\_\_str\_\_() 方法返回的字符串应该是一个可读的字符串表示，通常用于显示给用户。

```
>>> f = open('file.dat')
>>> f
<_io.TextIOWrapper name='file.dat' mode='r' encoding='UTF-8'>
>>>
```

\_\_str\_\_() 和 \_\_repr\_\_() 方法返回的字符串应该能够重新创建对象，即 eval(repr(x)) == x。\_\_str\_\_() 方法返回的字符串应该是一个可读的字符串表示，通常用于显示给用户。

format() 方法返回的字符串应该能够重新创建对象，即 eval(repr(x)) == x。\_\_str\_\_() 方法返回的字符串应该是一个可读的字符串表示，通常用于显示给用户。

%

## 8.2

```
format()

```

## 8.2.2 □□□□

\_\_format\_\_()  
format()

```
_formats = {  
    'ymd' : '{d.year}-{d.month}-{d.day}',  
    'mdy' : '{d.month}/{d.day}/{d.year}',  
    'dmy' : '{d.day}/{d.month}/{d.year}'  
}
```

**class Date**

```
:  
    def  
  
__init__(self, year, month, day):  
    self.year = year  
    self.month = month  
    self.day = day
```

**def**

```
__format__(self, code):  
    if  
  
code == '':  
        code = 'ymd'  
        fmt = _formats[code]  
    return
```

```
fmt.format(d=self)
```

Date

```
>>> d = Date(2012, 12, 21)  
>>> format(d)  
'2012-12-21'  
>>> format(d, 'mdy')  
'12/21/2012'  
>>> 'The date is {:ymd}'.format(d)
```

```
'The date is 2012-12-21'
>>> 'The date is {:mdy}'.format(d)
'The date is 12/21/2012'
>>>
```

## 8.2.3 日期

`__format__()` 是 Python 中用于格式化的方法。它允许你自定义对象的字符串表示。在 Python 中，日期和时间对象是 `datetime` 模块的一部分。你可以使用 `datetime` 模块来创建日期和时间对象，并使用 `__format__()` 方法来格式化它们。

```
>>> from datetime import
date
>>> d = date(2012, 12, 21)
>>> format(d)
'2012-12-21'
>>> format(d, '%A, %B %d, %Y')
'Friday, December 21, 2012'
>>> 'The end is {:%d %b %Y}. Goodbye'.format(d)
'The end is 21 Dec 2012. Goodbye'
>>>
```

有关字符串格式化的更多信息，请访问 <http://docs.python.org/3/library/string.html>。

## 8.3 字符串格式化

## 8.3.1

context-management protocol with

## 8.3.2

with\_\_enter\_\_()\_\_exit\_\_()

```
from socket import
socket, AF_INET, SOCK_STREAM

class LazyConnection:
:
    def
__init__(self, address, family=AF_INET, type=SOCK_STREAM):
    self.address = address
    self.family = AF_INET
    self.type = SOCK_STREAM
    self.sock = None

    def
__enter__(self):
    if
self.sock is not
None:
        raise RuntimeError
('Already connected')
```





```
' )
    resp = b''.join(iter(partial(s.recv, 8192), b''))
    # conn.__exit__() executes: connection closed
```

### 8.3.3 with

with 语句是 Python 2.5 引入的，它提供了一种简洁的方式来处理资源管理。with 语句通常用于打开文件、打开数据库连接、打开网络套接字等。with 语句的语法如下：

```
with 表达式 as 变量:
    语句块
```

with 语句的表达式必须实现上下文管理器接口。上下文管理器接口是一个由 `__enter__()` 和 `__exit__()` 方法组成的接口。with 语句在遇到 `with` 关键字时，会调用 `__enter__()` 方法，并将返回的值赋给 `as` 后面的变量。然后，with 语句会执行 `as` 后面的变量所指向的对象的 `__exit__()` 方法。如果 `__exit__()` 方法返回 `True`，则 with 语句会正常结束；如果返回 `False`，则 with 语句会抛出异常。

LazyConnection 是一个上下文管理器，它用于打开数据库连接。LazyConnection 的语法如下：

```
with LazyConnection(socket=socket, host=host, port=port, user=user, password=password) as conn:
    语句块
```

```

from socket import
socket, AF_INET, SOCK_STREAM

class LazyConnection
:
    def
__init__(self, address, family=AF_INET, type=SOCK_STREAM):
    self.address = address
    self.family = AF_INET
    self.type = SOCK_STREAM
    self.connections = []

    def
__enter__(self):
    sock = socket(self.family, self.type)
    sock.connect(self.address)
    self.connections.append(sock)
    return
sock

    def
__exit__(self, exc_ty, exc_val, tb):
    self.connections.pop().close()

# Example use

from functools import
partial

conn = LazyConnection(('www.python.org', 80))
with
conn as
s1:
    ...

```

```

    with
conn as
s2:
    ...
    # s1 and s2 are independent sockets

```

LazyConnection

`__enter__()`

`__exit__()`

with

`__enter__()`

`__exit__()`

with

contextmanager

9.22

12.6

## 8.4

## 8.4.1 类

Python 中，类（class）是对象的模板，它定义了对象的属性和方法。

## 8.4.2 类属性

类属性是类中定义的属性，它们属于类本身，而不是属于类的实例。

\_\_slot\_\_ 是类的一个属性，它用于指定类的槽（slots）。

```
class Date:
    __slots__ = ['year', 'month', 'day']
    def __init__(self, year, month, day):
        self.year = year
        self.month = month
        self.day = day
```

\_\_slots\_\_ 是 Python 中一个特殊的类属性，它用于限制类的实例只能拥有哪些属性。在 Python 中，每个类都有一个 \_\_dict\_\_ 属性，它是一个字典，用于存储类的属性。但是，如果类定义了 \_\_slots\_\_，那么类的实例只能拥有 \_\_slots\_\_ 中指定的属性，而不能拥有其他属性。这可以节省内存，并提高程序的性能。在 Python 3.6 之前，\_\_slots\_\_ 只能用于限制实例的属性，而不能用于限制类的属性。从 Python 3.6 开始，\_\_slots\_\_ 也可以用于限制类的属性。但是，\_\_slots\_\_ 只能用于限制实例的属性，而不能用于限制类的属性。在 Python 3.6 之前，\_\_slots\_\_ 只能用于限制实例的属性，而不能用于限制类的属性。从 Python 3.6 开始，\_\_slots\_\_ 也可以用于限制类的属性。但是，\_\_slots\_\_ 只能用于限制实例的属性，而不能用于限制类的属性。

## 8.4.3 类方法



## 8.5.2 類別

類別是Python中一個非常重要的概念，它允許我們將具有相同行為的對象組織在一起。類別定義了一個對象的藍圖，包括它的屬性（attributes）和方法（methods）。在Python中，類別是用大寫字母開頭的，並且通常跟隨著一個單詞的命名約定。

```
class A
:
    def
__init__(self):
    self._internal = 0          # An internal attribute

    self.public = 1            # A public attribute

    def
public_method(self):
    '''

    A public method

    '''

    ...

    def
    _internal_method(self):
        ...
```

Python은 객체 지향 프로그래밍 언어이다. 객체 지향 프로그래밍은 데이터를 객체로 표현하고, 객체 간의 상호작용을 통해 프로그램을 작성하는 방식이다. Python은 객체 지향 프로그래밍을 위한 다양한 라이브러리를 제공한다. 예를 들어, socket 라이브러리를 사용하여 네트워크 통신을 할 수 있다. 또한, sys.\_getframe() 함수를 사용하여 현재 호출 프레임의 정보를 얻을 수 있다.

다음은 Python에서 클래스와 메서드를 정의하는 예제이다.

```
class B
:
    def
__init__(self):
    self.__private = 0
    def
__private_method(self):
    ...
    def
public_method(self):
    ...
    self.__private_method()
    ...
```

Python에서는 클래스의 속성이나 메서드를 private로 선언할 수 있다. private로 선언된 속성이나 메서드는 클래스 외부에서 접근할 수 없다. 예를 들어, B 클래스의 \_\_private 속성은 클래스 외부에서 접근할 수 없다. 또한, \_\_private\_method 메서드도 클래스 외부에서 접근할 수 없다. 이는 Python의 이름 매נג글링(mangling) 규칙 때문이다. 이름 매נג글링은 클래스의 private 속성이나 메서드를 자동으로 \_\_classname\_\_로 변경하는 규칙이다. 예를 들어, B 클래스의 \_\_private 속성은 \_\_B\_\_private로 변경된다. 이는 Python의 이름 매נג글링 규칙 때문이다.



```

class C
(B):
    def
__init__(self):
    super().__init__()
    self.__private = 1          # Does not override
    B.__private

    # Does not override B.__private_method()

    def
__private_method(self):
    ...

```

```

    __private__private_method__
__C__private__C__private_method__
B

```

### 8.5.3

“”

```
lambda_ = 2.0      # Trailing _ to avoid clash with lambda
keyword
```

## 8.6

### 8.6.1 ☐ ☐

## 8.6.2 ☐☐☐☐

```
[2] property
property
```

```
class Person
```

```
:
    def
__init__(self, first_name):
    self.first_name = first_name

    # Getter function

    @property
    def
first_name(self):
    return
self._first_name

    # Setter function

    @first_name.setter
    def
first_name(self, value):
    if not
instance(value, str):
        raise TypeError
('Expected a string')
        self._first_name = value

    # Deleter function (optional)

    @first_name.deleter
    def
first_name(self):
        raise AttributeError
("Can't delete attribute")
```

```

class Person:
    def __init__(self, first_name):
        self.first_name = first_name

    @property
    def first_name(self):
        return self._first_name

    @first_name.setter
    def first_name(self, value):
        if not isinstance(value, str):
            raise TypeError('Expected a string')
        self._first_name = value

    @first_name.deleter
    def first_name(self):
        raise AttributeError("Can't delete attribute")

```

```

class Person:
    def __init__(self, first_name):
        self.first_name = first_name

    @property
    def first_name(self):
        return self._first_name

    @first_name.setter
    def first_name(self, value):
        if not isinstance(value, str):
            raise TypeError('Expected a string')
        self._first_name = value

    @first_name.deleter
    def first_name(self):
        raise AttributeError("Can't delete attribute")

```

```

>>> a = Person('Guido')
>>> a.first_name           # Calls the getter

'Guido'
>>> a.first_name = 42      # Calls the setter

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "prop.py", line 14, in first_name
    raise TypeError

('Expected a string')
TypeError: Expected a string
>>> del a.first_name

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: can't delete attribute
>>>

```

property getter/setter  
\_first\_name  
\_\_init\_\_() self.first\_name  
self.\_first\_name property  
\_\_init\_\_() self.first\_name  
setter  
self.first\_name self.\_first\_name  
getter/setter  
property

```
class Person:
    def __init__(self, first_name):
        self.set_first_name(first_name)

    # Getter function
    def get_first_name(self):
        return self._first_name

    # Setter function
    def set_first_name(self, first_name):
```



```
>>>
```

property decorator fget fset

property Java getter setter

```
class Person
:
    def
__init__(self, first_name):
    self.first_name = name
@property
def
first_name(self):
    return
self._first_name
@first_name.setter
def
first_name(self, value):
    self._first_name = value
```

property decorator

property decorator  
property decorator  
property decorator  
property decorator

property decorator  
property decorator

```
import math

class Circle:
    def
__init__(self, radius):
    self.radius = radius
    @property
    def
area(self):
    return
math.pi * self.radius ** 2
    @property
    def
perimeter(self):
    return
2 * math.pi * self.radius
```

property decorator  
radius area perimeter



□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> c = Circle(4.0)
>>> c.radius
4.0
>>> c.area                # □□□□□□()

50.26548245743669
>>> c.perimeter          # □□□□□□()

25.132741228718345
>>>
```

□□property□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□getter□setter□□□□□□□□

```
>>> p = Person('Guido')
>>> p.get_first_name()
'Guido'
>>> p.set_first_name('Larry')
>>>
```

□□□□□□□□□□Python□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□  
□□□□□RPC□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□get/set□□□□□□□□□□□□□□□□□□  
property□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□property  
□□□□□□□□□□

```

class Person
:
    def
__init__(self, first_name, last_name):
    self.first_name = first_name
    self.last_name = last_name

    @property
    def
first_name(self):
    return
self._first_name

    @first_name.setter
    def
first_name(self, value):
    if not
instance(value, str):
        raise TypeError
('Expected a string')
        self._first_name = value

        # Repeated property code, but for a different name (bad!)

    @property
    def
last_name(self):
    return
self._last_name

    @last_name.setter
    def
last_name(self, value):

```

```
        if not
instance(value, str):
            raise TypeError

('Expected a string')
self._last_name = value
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□8.9□□  
9.21□□

## 8.7 □□□□□□□□

### 8.7.1 □□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

### 8.7.2 □□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□super()□□□□□□□□  
□□□□□

```
class A
:
    def
spam(self):
    print
```

```

('A.spam')

class B
(A):
    def
spam(self):
    print
('B.spam')
    super().spam()    # Call parent spam()

```

super()은 부모 클래스의 \_\_init\_\_() 메서드를 호출하여 부모 클래스의 초기화 상태를 유지합니다.

```

class A
:
    def
__init__(self):
    self.x = 0

class B
(A):
    def
__init__(self):
    super().__init__()
    self.y = 1

```

이 코드는 Python의 클래스 상속을 보여주는 예제입니다.

```

class Proxy
:
    def
__init__(self, obj):
    self._obj = obj

    # Delegate attribute lookup to internal obj

    def
__getattr__(self, name):
    return
getattr(self._obj, name)

    # Delegate attribute assignment

    def
__setattr__(self, name, value):
    if
name.startswith('_'):
        super().__setattr__(name, value)    # Call
original __setattr__

    else

:
    setattr(self._obj, name, value)

```

```

    def __setattr__(self, value):
        if not value.startswith('_'):
            super().__setattr__(value)
        else:
            setattr(self._obj, value)

```

super()은 super class의 \_\_init\_\_을 호출하는 역할을 한다.  
\_\_init\_\_은 초기화 함수이다.

## 8.7.3 예제

super()은 Python의 class hierarchy를 따라 올라가는 역할을 한다.  
\_\_init\_\_은 class의 초기화 함수이다.

```
class Base:
    def __init__(self):
        print('Base.__init__')

class A(Base):
    def __init__(self):
        Base.__init__(self)
        print('A.__init__')
```

\_\_init\_\_은 class의 초기화 함수이다.  
\_\_init\_\_은 class의 초기화 함수이다.

```
class Base
:
    def
__init__(self):
    print
('Base.__init__')

class A
(Base):
    def
__init__(self):
    Base.__init__(self)
    print
('A.__init__')

class B
(Base):
    def
__init__(self):
    Base.__init__(self)
    print
('B.__init__')

class C
(A,B):
    def
__init__(self):
    A.__init__(self)
    B.__init__(self)
    print
('C.__init__')
```

---

□□□□□□□□□□□□□□Base.\_\_init\_\_()□□□□□□□□  
□□□□□□□□

```
>>> c = C()
Base.__init__
A.__init__
Base.__init__
B.__init__
C.__init__
>>>
```

□□□□□□Base.\_\_init\_\_()□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□super()□□□□□□□□  
□□□□□□□□

```
class Base
:
    def
__init__(self):
    print
('Base.__init__')

class A
(Base):
    def
__init__(self):
    super().__init__()
    print
('A.__init__')
```



```

class B
(Base):
    def
__init__(self):
    super().__init__()
    print
('B.__init__')

class C
(A,B):
    def
__init__(self):
    super().__init__() # Only one call to super() here

    print
('C.__init__')

```

1. Python 2.7 中，\_\_init\_\_() 方法在类创建时调用。

```

>>> c = C()
Base.__init__
B.__init__
A.__init__
C.__init__
>>>

```

2. Python 3.0 中，\_\_init\_\_() 方法在类创建时调用。

Python MRO [3] Python MRO 是什么？  
如何查看？

```
>>> C.__mro__  
(<class '__main__.C'>, <class '__main__.A'>, <class  
'__main__.B'>,  
<class '__main__.Base'>, <class 'object'>)  
>>>
```

Python MRO 是什么？  
如何查看？

MRO 是什么？  
C3 Linearization 是什么？  
3 是什么？

- 如何查看？
- 如何查看 MRO？
- 如何查看 MRO？

MRO 是什么？  
class hierarchy

super() Python MRO 是什么？  
如何查看？

`super()` 返回的对象的 MRO 和  
当前对象的 MRO 是一样的。  
`Base.__init__()` 返回 None。

在 `super()` 返回的对象的 MRO 中，  
当前对象排在第一位。  
例如：

```
class A:
    def
spam(self):
    print
('A.spam')

super().spam()
```

运行结果如下：

```
>>> a = A()
>>> a.spam()
A.spam
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 4, in spam
AttributeError: 'super' object has no attribute 'spam'
>>>
```

□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> class B
:
...     def
spam(self):
...         print
('B.spam')
...

>>> class C
(A,B):
...     pass

...

>>> c = C()
>>> c.spam()
A.spam
B.spam
>>>
```

□□□□□□□□□□A□□□□super().spam()□□□□□□  
□□□□B□□spam()□□——B□A□□□□□□□□□□□□□□  
□□C□MRO□□□□□□□□

```
>>> C.__mro__
(<class '__main__.C'>, <class '__main__.A'>, <class
 '__main__.B'>,
<class 'object'>)
>>>
```

---

Mixin class 8.13-8.18

super() 8.13-8.18

Python super() Raymond Hettinger “Python’s super() considered Super!” [4]

## 8.8

### 8.8.1

### 8.8.2

□□□□□□□□□□□□□□□□□□□□name□

```
class Person
:
    def
__init__(self, name):

self.name = name

# Getter function

@property
    def
name(self):
    return
self._name

# Setter function

@name.setter
    def
name(self, value):
    if not
instance(value, str):
        raise TypeError
```

```

('Expected a string')

self._name = value

# Deleter function

@name.deleter
def
name(self):
    raise AttributeError

("Can't delete attribute")

```

class Person:
 name

```

class SubPerson
(Person):

@property
def
name(self):
    print
('Getting name')
    return
super().name

```

```

@name.setter
    def
name(self, value):
    print
('Setting name to', value)

super(SubPerson, SubPerson).name.__set__(self, value)

@name.deleter
    def
name(self):
    print
('Deleting name')

super(SubPerson, SubPerson).name.__delete__(self)

```

□□□□□□□□□□□□□□

```

>>> s = SubPerson('Guido')
Setting name to Guido
>>> s.name
Getting name
'Guido'
>>> s.name = 'Larry'
Setting name to Larry
>>> s.name = 42
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "example.py", line 16, in name
    raise TypeError
('Expected a string')

```



```
TypeError: Expected a string
>>>
```

[illegible]

```
class SubPerson
(Person):

@Person.name.getter
    def

name(self):
    print

    ('Getting name')
    return

super().name
```

setter

```
class SubPerson
(Person):

    @Person.name.setter
    def
name(self, value):
    print
('Setting name to', value)
```

```
super(SubPerson, SubPerson).name.__set__(self, value)
```

### 8.8.3 属性

getter setter deleter 属性描述符  
属性描述符是 Python 中用于管理属性访问的类。它们定义了如何从对象中获取、设置或删除属性。属性描述符通常用于实现类属性、只读属性、私有属性等。

在 Python 中，属性描述符是一个类，它实现了 `__get__`、`__set__` 和 `__delete__` 方法。这些方法分别用于从对象中获取、设置或删除属性。属性描述符通常用于实现类属性、只读属性、私有属性等。

```
class SubPerson(Person):  
    @property  
    def name(self):  
        return super(SubPerson, SubPerson).name.__  
            set__(self, value)  
    @name.setter  
    def name(self, value):  
        super(SubPerson, SubPerson).name.__  
            set__(self, value)
```

在 Python 中，属性描述符是一个类，它实现了 `__get__`、`__set__` 和 `__delete__` 方法。这些方法分别用于从对象中获取、设置或删除属性。属性描述符通常用于实现类属性、只读属性、私有属性等。

```
class SubPerson
```

```
(Person):
```

```
@property
```

```
# Doesn't work
```

```
def
name(self):
    print
('Getting name')
    return
super().name
```

setter

```
>>>
s = SubPerson('Guido')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "example.py", line 5, in __init__
self.name = name
AttributeError: can't set attribute
>>>
```

```
class SubPerson
(Person):
@Person.getter
def
name(self):
    print
```

```
('Getting name')
    return

super().name
```

getter

```
>>>

s = SubPerson('Guido')
>>>

s.name
Getting name
'Guido'
>>>

s.name = 'Larry'
>>>

s.name
Getting name
'Larry'
>>>

s.name = 42
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "example.py", line 16, in name
    raise

TypeError('Expected a string')
TypeError: Expected a string
>>>
```

```
class Person:
    def __init__(self):
        super().__init__()
```

8.9

```
# A descriptor

class String:
    def
    __init__(self, name):

self.name = name

    def
    __get__(self, instance, cls):
        if
instance is None:
            return

self
        return

instance.__dict__[self.name]

    def
    __set__(self, instance, value):
        if not

isinstance(value, str):
```

```
        raise TypeError
('Expected a string')

instance.__dict__[self.name] = value
# A class with a descriptor

class Person
:

name = String('name')
    def
        __init__(self, name):

self.name = name
# Extending a descriptor with a property

class SubPerson
(Person):

@property
    def
        name(self):
            print
('Getting name')
            return
super().name

@name.setter
```

```

    def
    name(self, value):
        print
    ('Setting name to', value)

    super(SubPerson, SubPerson).name.__set__(self, value)

    @name.deleter
    def
    name(self):
        print
    ('Deleting name')

    super(SubPerson, SubPerson).name.__delete__(self)

```

setter deleter Python bug  
<http://bugs.python.org/issue14965>  
 Python

## 8.9

### 8.9.1

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□

## 8.9.2 □□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□

```
# Descriptor attribute for an integer type-checked attribute

class Integer
:
    def
    __init__(self, name):
self.name = name
    def
    __get__(self, instance, cls):
        if
instance is None:
            return
self
        else
:
            return
instance.__dict__[self.name]
    def
```



```

__set__(self, instance, value):
    if not
        isinstance(value, int):
            raise TypeError
                ('Expected an int')

instance.__dict__[self.name] = value

    def
        __delete__(self, instance):
            del
                instance.__dict__[self.name]

```

1. 实现 \_\_get\_\_() 和 \_\_set\_\_() 方法  
 2. 实现 \_\_delete\_\_() 方法  
 3. 实现 \_\_get\_\_() 方法  
 4. 实现 \_\_set\_\_() 方法  
 5. 实现 \_\_delete\_\_() 方法

1. 实现 \_\_get\_\_() 方法  
 2. 实现 \_\_set\_\_() 方法  
 3. 实现 \_\_delete\_\_() 方法

```

class Point
:
    x = Integer('x')
    y = Integer('y')
    def
        __init__(self, x, y):
            self.x = x

```

```
self.y = y
```

```
    """
    __get__(), __set__(), __delete__()
    """
```

```
>>>
p = Point(2, 3)
>>>
p.x                # Calls Point.x.__get__(p, Point)
2
>>>
p.y = 5            # Calls Point.y.__set__(p, 5)
>>>
p.x = 2.3          # Calls Point.x.__set__(p, 2.3)
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "descrip.py", line 12, in __set__
    raise TypeError
('Expected an int')
TypeError: Expected an int
>>>
```

```
    """
    __dict__
    """
```

self.name  
[...]

### 8.9.3

Python  
@classmethod  
@staticmethod  
@property  
\_\_slots\_\_

get  
set  
delete  
[...]

[...]

*# Does NOT work*

**class Point**

:

**def**

**\_\_init\_\_(self, x, y):**

self.x = Integer('x') *# No! Must be a class variable*

self.y = Integer('y')

self.x = x

```
self.y = y
```

```
__get__()
```

```
# Descriptor attribute for an integer type-checked attribute
```

```
class Integer
```

```
def
```

```
__get__(self, instance, cls):
    if
```

```
instance is
```

|       |  |
|-------|--|
| None: |  |
|-------|--|

```

return

```

```
self
```

```

else

```

```

return

```

```
instance.__dict__[self.name]
```

■ ■ ■

```
__get__() [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] []
```

```

instance

```

None

[illegible]

```
>>> p = Point(2,3)
>>> p.x          # Calls Point.x.__get__(p, Point)

2
>>> Point.x      # Calls Point.x.__get__(None, Point)

<__main__.Integer object at 0x100671890>
>>>
```

```
# Descriptor for a type-checked attribute

class Typed
:
    def

__init__(self, name, expected_type):
    self.name = name
    self.expected_type = expected_type

    def

__get__(self, instance, cls):
    if
instance is
None:
        return

    self

    else
```

```

:
    return
instance.__dict__[self.name]

    def
__set__(self, instance, value):
    if not
isinstance(value, self.expected_type):
    raise TypeError

    ('Expected ' + str(self.expected_type))
    instance.__dict__[self.name] = value

    def
__delete__(self, instance):
    del
instance.__dict__[self.name]
# Class decorator that applies it to selected attributes

def
typeassert(**kwargs):
    def
decorate(cls):
        for
name, expected_type in
kwargs.items():
        # Attach a Typed descriptor to the class

        setattr(cls, name, Typed(name, expected_type))
    return
cls

```

```

    return

decorate

# Example use

@typeassert(name=str, shares=int, price=float)
class Stock

:
    def

__init__(self, name, shares, price):
    self.name = name
    self.shares = shares
    self.price = price

```

8.10 8.10.1 8.10.1

## 8.10 8.10.1 8.10.1

### 8.10.1 8.10.1

8.10.1 8.10.1 8.10.1

## 8.10.2 装饰器

装饰器（decorator）

```
class lazyproperty
:
    def
__init__(self, func):
    self.func = func
    def
__get__(self, instance, cls):
    if
instance is
None:
        return
self
    else
:
        value = self.func(instance)
        setattr(instance, self.func.__name__, value)
        return
value
```

装饰器（decorator）

```
import math
```



```

class Circle
:
    def
__init__(self, radius):
    self.radius = radius

    @lazyproperty
    def
area(self):
    print
('Computing area')
    return
math.pi * self.radius ** 2

    @lazyproperty
    def
perimeter(self):
    print
('Computing perimeter')
    return
2 * math.pi * self.radius

```

□□□□□□□□□□□□□□□□□□

```

>>> c = Circle(4.0)
>>> c.radius
4.0
>>> c.area
Computing area
50.26548245743669
>>> c.area
50.26548245743669
>>> c.perimeter

```

```
Computing perimeter
25.132741228718345
>>> c.perimeter
25.132741228718345
>>>
```

□□□□□□□“Computing  
area”□“Computing perimeter”□□□□□□□

### 8.10.3 ☐ ☐

**8.9**

`__get__()` `__set__()` `__delete__()`

`__get__()`

much weaker binding

`get ()`

```

    @property
    def lazyproperty(self):
        def _get_():
            # ...
        return lazyproperty(_get_)

```



```
>>> c.area = 25
>>> c.area
25
>>>
```

```
def
lazyproperty(func):
    name = '_lazy_' + func.__name__
    @property
    def
lazy(self):
    if
hasattr(self, name):
    return
getattr(self, name)
else
:
    value = func(self)
    setattr(self, name, value)
    return
value
return
lazy
```

set

```

>>> c = Circle(4.0)
>>> c.area
Computing area
50.26548245743669
>>> c.area
50.26548245743669
>>> c.area = 25
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: can't set attribute
>>>

```

1. 使用 `get` 方法  
 2. 使用 `getter` 方法

3. 使用 `property` 方法  
 4. 使用 `8.6` 和 `8.9`

## 8.11 使用 `__init__` 方法

### 8.11.1 使用 `__init__` 方法

1. 使用 `__init__` 方法  
 2. 使用 `__init__` 方法

### 8.11.2 使用 `__init__` 方法

1. 使用 `__init__` 方法  
 2. 使用 `__init__` 方法

```

class Structure
:
    # Class variable that specifies expected fields

    _fields= []
    def
__init__(self, *args):
    if
len(args) != len(self._fields):
        raise TypeError
('Expected {} arguments'.format(len(self._fields)))

        # Set the arguments

        for
name, value in
zip(self._fields, args):
        setattr(self, name, value)

        # Example class definitions

        if
__name__ == '__main__':
        class Stock
(Structure):
            _fields = ['name', 'shares', 'price']

            class Point
(Structure):
            _fields = ['x', 'y']

            class Circle

```

```
(Structure):
    _fields = ['radius']
    def
area(self):
    return
math.pi * self.radius ** 2
```

[illegible]

```
>>> s = Stock('ACME', 50, 91.1)
>>> p = Point(2, 3)
>>> c = Circle(4.5)
>>> s2 = Stock('ACME', 50)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "structure.py", line 6, in __init__
    raise TypeError
('Expected {} arguments'.format(len(self._fields)))
TypeError: Expected 3 arguments
```

```

    struct {
        int _fields[ ];
    };

```

```
class Structure
:
    _fields= []
    def
__init__(self, *args, **kwargs):
    if
```

```

len(args) > len(self._fields):
    raise TypeError

('Expected {} arguments'.format(len(self._fields)))

    # Set all of the positional arguments

    for
name, value in
zip(self._fields, args):
    setattr(self, name, value)

    # Set the remaining keyword arguments

    for
name in
self._fields[len(args):]:
    setattr(self, name, kwargs.pop(name))

    # Check for any remaining unknown arguments

    if
kwargs:
    raise TypeError

('Invalid argument(s): {}'.format(','.join(kwargs)))

# Example use

if
__name__ == '__main__':
    class Stock

(Structure):

```



```
_fields = ['name', 'shares', 'price']

s1 = Stock('ACME', 50, 91.1)
s2 = Stock('ACME', 50, price=91.1)
s3 = Stock('ACME', shares=50, price=91.1)
```

[illegible]

```

class Structure
:
    # Class variable that specifies expected fields

    _fields= []
    def

__init__(self, *args, **kwargs):
    if

len(args) != len(self._fields):
    raise TypeError

('Expected {} arguments'.format(len(self._fields)))

    # Set the arguments

    for

name, value in

zip(self._fields, args):
    setattr(self, name, value)

    # Set the additional arguments (if any)

    extra_args = kwargs.keys() - self._fields

```

```

        for
name in
extra_args:
    setattr(self, name, kwargs.pop(name))
    if
kwargs:
        raise TypeError
('Duplicate values for {}'.format(','.join(kwargs)))
# Example use

if
__name__ == '__main__':
    class Stock
(Structure):
    _fields = ['name', 'shares', 'price']

    s1 = Stock('ACME', 50, 91.1)
    s2 = Stock('ACME', 50, 91.1, date='8/2/2012')

```

## 8.11.3

```

__init__()
__init__()

```

```

class Stock
:
    def

```

```
__init__(self, name, shares, price):
    self.name = name
    self.shares = shares
    self.price = price
```

```
class Point
```

```
:
```

```
    def
```

```
__init__(self, x, y):
    self.x = x
    self.y = y
```

```
class Circle
```

```
:
```

```
    def
```

```
__init__(self, radius):
    self.radius = radius
```

```
    def
```

```
area(self):
    return
```

```
math.pi * self.radius ** 2
```

setattr()

```
class Structure
```

```
:
```

```
    # Class variable that specifies expected fields
```

```
    _fields= []
```

```
    def
```

```

__init__(self, *args):
    if
len(args) != len(self._fields):
        raise TypeError

('Expected {} arguments'.format(len(self._fields)))

    # Set the arguments (alternate)

self.__dict__.update(zip(self._fields,args))

```

```

class Stock(Structure):
    __slots__ = ('property',)
    # ...

```

```

class Stock(Structure):
    IDE
    # ...

```

```

>>> help(Stock)
Help on class Stock in module __main__:

class Stock(Structure)
...

| Methods inherited from Structure:
|
| __init__(self, *args, **kwargs)
|
...

```

```
>>>
```

\_\_init\_\_() 9.16

“frame hack”

```
def
init_fromlocals(self):
    import sys

    locs = sys._getframe(1).f_locals
    for
k, v in
locs.items():
    if
k != 'self':
        setattr(self, k, v)
class Stock
:
    def
__init__(self, name, shares, price):
    init_fromlocals(self)
```

init\_fromlocals() sys.\_getframe() \_\_init\_\_()

\_\_init\_\_()  
IDE  
50%  
[...]

## 8.12 [...]

### 8.12.1 [...]

[...]

### 8.12.2 [...]

abc[...]

```
from abc import
ABCMeta, abstractmethod

class IStream
(metaclass=ABCMeta):
    @abstractmethod
    def
read(self, maxbytes=-1):
    pass
```

```
@abstractmethod
def
write(self, data):
    pass
```

```

    """
    """
```

```
a = IStream()      # TypeError: Can't instantiate abstract
class

                    # IStream with abstract methods read, write
```

```

    """
    """
```

```
class SocketStream

(IStream):
    def

    read(self, maxbytes=-1):
        ...
    def

    write(self, data):
        ...
```

Interface for streams  
IStream  
A stream is an object that implements the IStream interface

```
def  
serialize(obj, stream):  
    if not  
instance(stream, IStream):  
    raise TypeError  
( 'Expected an IStream' )  
    ...
```

Example of a stream  
A stream is an object that implements the IStream interface  
A stream is an object that implements the IStream interface  
A stream is an object that implements the IStream interface

```
import io  
  
# Register the built-in I/O classes as supporting our  
interface  
  
IStream.register(io.IOBase)  
  
# Open a normal file and type check  
  
f = open('foo.txt')  
instance(f, IStream)      # Returns True
```



```

    @abstractmethod
    @property
    @abstractmethod

```

```

from abc import
    ABCMeta, abstractmethod

class A
    (metaclass=ABCMeta):
        @property
        @abstractmethod
        def

    name(self):
        pass

        @name.setter
        @abstractmethod
        def

    name(self, value):
        pass

        @classmethod
        @abstractmethod
        def

    method1(cls):
        pass

```



```

if
instance(x, collections.Iterable):
    ...
# Check if x has a size

if
instance(x, collections.Sized):
    ...
# Check if x is a mapping

if
instance(x, collections.Mapping):
    ...

```

```

    □□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□

```

```

from decimal import
Decimal
import numbers

x = Decimal('3.4')
instance(x, numbers.Real)      # Returns False

```

```
class Base34(Base):
    """Base class for Base34. Uses a descriptor to set a value"""
    def __init__(self, value):
        self._value = value
```

```
class Python(Base):
    """Python class. Uses a descriptor to set a value"""
    def __init__(self, value):
        self._value = value
```

## 8.13 Base class. Uses a descriptor to set a value

### 8.13.1 Base

```
class Base:
    """Base class. Uses a descriptor to set a value"""
    def __init__(self, value):
        self._value = value
```

### 8.13.2 Python

```
class Python(Base):
    """Python class. Uses a descriptor to set a value"""
    def __init__(self, value):
        self._value = value
```

```
class Python(Base):
    """Python class. Uses a descriptor to set a value"""
```

```
# Base class. Uses a descriptor to set a value
```

```

class Descriptor
:
    def
__init__(self, name=None, **opts):
    self.name = name
    for
key, value in
opts.items():
    setattr(self, key, value)

    def
__set__(self, instance, value):
    instance.__dict__[self.name] = value

# Descriptor for enforcing types

class Typed
(Descriptor):
    expected_type = type(None)

    def
__set__(self, instance, value):
        if not
isinstance(value, self.expected_type):
            raise TypeError
('expected ' + str(self.expected_type))
        super().__set__(instance, value)

# Descriptor for enforcing values

class Unsigned

```

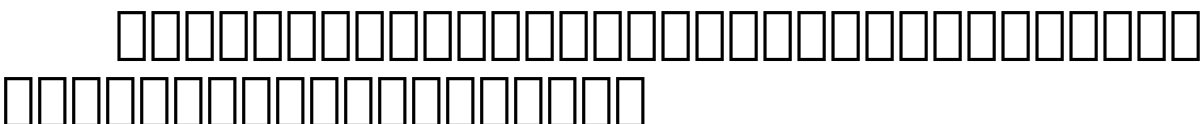
```

(Descriptor):
    def
__set__(self, instance, value):
    if
value < 0:
        raise ValueError
('Expected >= 0')
    super().__set__(instance, value)

class MaxSized
(Descriptor):
    def
__init__(self, name=None, **opts):
    if
'size' not in
opts:
        raise TypeError
('missing size option')
    super().__init__(name, **opts)

    def
__set__(self, instance, value):
    if
len(value) >= self.size:
        raise ValueError
('size must be < ' + str(self.size))
    super().__set__(instance, value)

```



```

class Integer
(Typed):
    expected_type = int

class UnsignedInteger
(Integer, Unsigned):
    pass

class Float
(Typed):
    expected_type = float

class UnsignedFloat
(Float, Unsigned):
    pass

class String
(Typed):
    expected_type = str

class SizedString
(String, MaxSized):
    pass

```

□□□□□□□□□□□□□□□□□□□□□□

```

class Stock

```

```

:
    # Specify constraints

    name = SizedString('name',size=8)
    shares = UnsignedInteger('shares')
    price = UnsignedFloat('price')
    def

__init__(self, name, shares, price):
    self.name = name
    self.shares = shares
    self.price = price

```

```

    """
    """

```

```

>>> s = Stock('ACME', 50, 91.1)
>>> s.name
'ACME'
>>> s.shares = 75
>>> s.shares = -10
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "example.py", line 17, in __set__
    super().__set__(instance, value)
  File "example.py", line 23, in __set__
    raise ValueError

('Expected >= 0')
ValueError: Expected >= 0
>>> s.price = 'a lot'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "example.py", line 16, in __set__
    raise TypeError

('expected ' + str(self.expected_type))
TypeError: expected <class 'float'>
>>> s.name = 'ABRACADABRA'

```





```

        return

decorate

# Example

@check_attributes(name=SizeString(size=8),
                  shares=UnsignedInteger,
                  price=UnsignedFloat)
class Stock
:
    def

__init__(self, name, shares, price):
    self.name = name
    self.shares = shares
    self.price = price

```

□□□□□□□□□□□□□□□□

```

# A metaclass that applies checking

class checkedmeta
(type):
    def

__new__(cls, clsname, bases, methods):
    # Attach attribute names to the descriptors

    for

key, value in
methods.items():
    if

```

```

isinstance(value, Descriptor):
    value.name = key
    return

type.__new__(cls, clsname, bases, methods)
# Example

class Stock

(metaclass=checkedmeta):
    name = SizedString(size=8)
    shares = UnsignedInteger()
    price = UnsignedFloat()
    def

__init__(self, name, shares, price):
    self.name = name
    self.shares = shares
    self.price = price

```

### 8.13.3

mixin super()  
 8.9 8.18 9.12  
 9.19

Descriptor \_\_set\_\_() \_\_get\_\_()

```

    mixin
    UnsignedMaxSizedTyped
    ...

```

```

    __init__(**opts)MaxSized
    Descriptor
    mixin
    super()

```

```

type class Integer Float
String
Typed expected_type

```

```


```

```
# Normal
```

```
class Point
```

```
:
```

```
    x = Integer('x')
```

```
    y = Integer('y')
```

```
# Metaclass
```

```
(metaclass=checkedmeta):
    x = Integer()
    y = Integer()
```

A diagram showing a 5x25 grid of squares. The top row has 25 squares. The second row has 25 squares. The third row has 25 squares. The fourth row has 25 squares. The fifth row has 10 squares.

```
class Mixin:
    def super():
        pass
```

```

:
    def
__init__(self, name=None, **opts):
    self.name = name
    for
key, value in

```

```

    opts.items():
        setattr(self, key, value)

    def

__set__(self, instance, value):
    instance.__dict__[self.name] = value

# Decorator for applying type checking

def

Typed(expected_type, cls=None):
    if

    cls is

    None:
        return lambda

    cls: Typed(expected_type, cls)

        super_set = cls.__set__
        def

__set__(self, instance, value):
    if

    not

    isinstance(value, expected_type):
        raise TypeError

    ('expected ' + str(expected_type))
        super_set(self, instance, value)
        cls.__set__ = __set__
        return

    cls

# Decorator for unsigned values

def

```

```

Unsigned(cls):
    super_set = cls.__set__
    def

__set__(self, instance, value):
    if

value < 0:
        raise ValueError

('Expected >= 0')
    super_set(self, instance, value)
    cls.__set__ = __set__
    return

cls

# Decorator for allowing sized values

def

MaxSized(cls):
    super_init = cls.__init__
    def

__init__(self, name=None, **opts):
    if

'size' not in

opts:
        raise TypeError

('missing size option')
    super_init(self, name, **opts)
    cls.__init__ = __init__

    super_set = cls.__set__
    def

__set__(self, instance, value):
    if

```

```
len(value) >= self.size:
    raise ValueError

('size must be < ' + str(self.size))
    super_set(self, instance, value)
    cls.__set__ = __set__
    return

cls

# Specialized descriptors

@Typed(int)
class Integer

    (Descriptor):
        pass

@Unsigned
class UnsignedInteger

    (Integer):
        pass

@Typed(float)
class Float

    (Descriptor):
        pass

@Unsigned
class UnsignedFloat

    (Float):
        pass
```



```
@Typed(str)
class String

(Descriptor):
    pass

@MaxSized
class SizedString

(String):
    pass
```

100%
 mixin

# 8.14

## 8.14.1

100%

## 8.14.2

`collections` 模块包含 `collections.Iterable` 抽象基类，用于定义可迭代的对象。

```
import collections

class A
(collections.Iterable):
    pass
```

`collections.Iterable` 抽象基类要求子类实现 `__iter__` 方法。

```
>>>
a = A()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't instantiate abstract class A with abstract
methods __iter__
>>>
```

从 Python 4.2 到 4.7 版本，`__iter__` 方法是必需的。

`collections` 模块包含以下类：  
`Sequence` `MutableSequence` `Mapping`  
`MutableMapping` `Set` `MutableSet`  
`Container` `Iterable` `Sized` `Sequence`  
`MutableSequence` 类包含以下方法：  
`__getitem__` `__len__` `__iter__` `__reversed__`  
`__delitem__` `__setitem__` `__eq__` `__neq__`  
`__lt__` `__le__` `__gt__` `__ge__`

```
>>> import collections

>>>

collections.Sequence()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't instantiate abstract class Sequence with
abstract methods \
__getitem__, __len__
>>>
```

`Sequence` 类包含以下方法：  
`__getitem__` `__len__` `__iter__` `__reversed__`  
`__delitem__` `__setitem__` `__eq__` `__neq__`  
`__lt__` `__le__` `__gt__` `__ge__`

```
import collections

import bisect
```

```

class SortedItems
(collections.Sequence):
    def

    __init__(self, initial=None):
        self._items = sorted(initial) if initial is None else
[]

    # Required sequence methods

    def

    __getitem__(self, index):
        return
self._items[index]

    def

    __len__(self):
        return
len(self._items)

    # Method for adding an item in the right location

    def

    add(self, item):
        bisect.insort(self._items, item)

```

□□□□□□□□□□□□

```

>>>
items = SortedItems([5, 1, 3])
>>>

```

```
list(items)
[1, 3, 5]
>>>

items[0]
1
>>>

items[-1]
5
>>>

items.add(2)
>>>

list(items)
[1, 2, 3, 5]
>>>

items.add(-10)
>>>

list(items)
[-10, 1, 2, 3, 5]
>>>

items[1:4]
[1, 2, 3]
>>>

3 in
items
True
>>>

len(items)
5
>>>

for
n in
```

```

items:
...     print
(n)
...
-10
1
2
3
5
>>>

```

SortedItems()
 len()
 in

bisect
 bisect.insort()

## 8.14.3

collections

SortedItems

```

>>>
items = SortedItems()
>>> import collections

```

```
>>>
    isinstance(items, collections.Iterable)
True
>>>
    isinstance(items, collections.Sequence)
True
>>>
    isinstance(items, collections.Container)
True
>>>
    isinstance(items, collections.Sized)
True
>>>
    isinstance(items, collections.Mapping)
False
>>>
```

```
collections[]  
[]  
collections.MutableSequence[]
```

```
class Items
(collections.MutableSequence):
    def
        __init__(self, initial=None):
            self._items = list(initial) if
                initial is
                    None else
```

```

[]

    # Required sequence methods

    def
__getitem__(self, index):
    print
('Getting:', index)
    return
self._items[index]

    def
__setitem__(self, index, value):
    print
('Setting:', index, value)
    self._items[index] = value

    def
__delitem__(self, index):
    print
('Deleting:', index)
    del
self._items[index]

    def
insert(self, index, value):
    print
('Inserting:', index, value)
    self._items.insert(index, value)

    def
__len__(self):

```



```
        print
('Len')
        return

len(self._items)
```

□□□□□Items□□□□□□□□□□□□□□□□□□□□  
□□□append()□remove()□count()□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>>
a = Items([1, 2, 3])
>>>
len(a)
Len
3
>>>
a.append(4)
Len
Inserting: 3 4
>>>
a.append(2)
Len
Inserting: 4 2
>>>
a.count(2)
Getting: 0
Getting: 1
Getting: 2
Getting: 3
Getting: 4
Getting: 5
2
```

```
>>>  
  
    a.remove(3)  
Getting: 0  
Getting: 1  
Getting: 2  
Deleting: 2  
>>>
```

Python  
numbers  
8.12

## 8.15

### 8.15.1

delegate

### 8.15.2

```
class A  
:  
    def
```

```
spam(self, x):
    pass

    def
foo(self):
    pass

class B
:
    def
__init__(self):
    self._a = A()

    def
spam(self, x):
    # Delegate to the internal self._a instance

    return
self._a.spam(x)

    def
foo(self):
    # Delegate to the internal self._a instance

    return
self._a.foo()

    def
bar(self):
    pass
```

```
class A:
    def spam(self, x):
        pass

    def foo(self):
        pass

    def __getattr__(self, name):
        pass
```

```
class A:
    def spam(self, x):
        pass

    def foo(self):
        pass

class B:
    def __init__(self):
        self._a = A()

    def bar(self):
        pass
```

```

    # Expose all of the methods defined on class A

    def
__getattr__(self, name):
    return
getattr(self._a, name)

```

\_\_getattr\_\_() 方法在类 B 中定义，用于在类 B 中查找属性。如果属性不存在，则委托给类 A 的 \_\_getattr\_\_() 方法。

```

b = B()
b.bar()      # Calls B.bar() (exists on B)

b.spam(42)   # Calls B.__getattr__('spam') and delegates to A.spam

```

下面是一个简单的代理类示例。

```

# A proxy class that wraps around another object, but

# exposes its public attributes

class Proxy
:
    def

```

```

__init__(self, obj):
    self._obj = obj

    # Delegate attribute lookup to internal obj

    def

__getattr__(self, name):
    print

    ('getattr:', name)
    return

    getattr(self._obj, name)

    # Delegate attribute assignment

    def

__setattr__(self, name, value):
    if

    name.startswith('_'):
        super().__setattr__(name, value)
    else

:
        print

    ('setattr:', name, value)
    setattr(self._obj, name, value)

    # Delegate attribute deletion

    def

__delattr__(self, name):
    if

    name.startswith('_'):
        super().__delattr__(name)

```

```

        else
:
            print
('delattr:', name)
            delattr(self._obj, name)

```

[illegible]

```
class Spam
:
    def
__init__(self, x):
    self.x = x
    def
bar(self, y):
    print
('Spam.bar:', self.x, y)

# Create an instance

s = Spam(2)

# Create a proxy around it

p = Proxy(s)

# Access the proxy

print
(p.x)      # Outputs 2
```

```
p.x = 37      # Changes s.x to 37
```

```
p.x = 37      # Changes s.x to 37
```

### 8.15.3 ☐☐

```
class A
:
    def
spam(self, x):
    print
('A.spam', x)
    def
foo(self):
    print
('A.foo')
class B
(A):
    def
```



```
spam(self, x):  
    print  
  
('B.spam')  
    super().spam(x)  
  
    def  
bar(self):  
    print  
  
('B.bar')
```

□□□□□□□□□□□□□□□□

```
class A  
  
:  
    def  
spam(self, x):  
    print  
  
('A.spam', x)  
  
    def  
foo(self):  
    print  
  
('A.foo')  
  
class B  
  
:  
    def  
__init__(self):  
    self._a = A()  
  
    def
```

```

spam(self, x):
    print

('B.spam', x)
    self._a.spam(x)

    def

bar(self):
    print

('B.bar')

    def

__getattr__(self, name):
    return

getattr(self._a, name)

```

1. 在类 B 中定义一个方法 spam，它接收一个参数 x，并调用 self.\_a.spam(x)。

2. 在类 B 中定义一个方法 bar，它调用 self.\_a.spam(x)。

3. 在类 B 中定义一个方法 \_\_getattr\_\_，它调用 getattr(self.\_a, name)。

4. 在类 B 中定义一个方法 \_\_setattr\_\_，它调用 setattr(self.\_a, name, value)。

```

class ListLike
:
    def
__init__(self):
    self._items = []
    def
__getattr__(self, name):
    return
getattr(self._items, name)

```

ListLike
 append()
 insert()
 len()

```

>>> a = ListLike()
>>> a.append(2)
>>> a.insert(0, 1)
>>> a.sort()
>>> len(a)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: object of type 'ListLike' has no len()
>>> a[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'ListLike' object does not support indexing
>>>

```

```

class ListLike

```



[illegible]

## 8.16.1 ☐☐

```
__init__()
    """
```

## 8.16.2 ☐☐☐☐

```
import time

class Date:
    # Primary constructor

    def
__init__(self, year, month, day):
    self.year = year
    self.month = month
    self.day = day

    # Alternate constructor

    @classmethod
    def
```

```
today(cls):
    t = time.localtime()
    return

cls(t.tm_year, t.tm_mon, t.tm_mday)
```

Date.today() [] [] [] []

```
a = Date(2012, 12, 21)      # Primary
b = Date.today()           # Alternate
```

### 8.16.3 ☐☐

Diagram illustrating a sequence of tokens (represented by boxes) used in a model. The sequence consists of 32 tokens, with the 16th token being the special token 'cls'.

```
class NewDate

(Date):
    pass

c = Date.today()      # Creates an instance of Date (cls=Date)
```

```
d = NewDate.today()      # Creates an instance of NewDate
(cls=NewDate)
```

```
def __init__(self):
    """
    Constructor for NewDate class.
    """
```

```
def __init__(self):
    """
```

```
class Date
:
    def
__init__(self, *args):
    if
len(args) == 0:
        t = time.localtime()
        args = (t.tm_year, t.tm_mon, t.tm_mday)
        self.year, self.month, self.day = args
```

```
def __init__(self):
    """
    Constructor for Date class.
    """
```

```
a = Date(2012, 12, 21)      # Clear. A specific date.
```

```

b = Date()                                # ??? What does this do?

# Class method version

c = Date.today()                          # Clear. Today's date.

```

□□□□□□□□Date.today()□□□  
 Date.\_\_init\_\_()□□□□□□□□□□□□□□□□□□□□  
 Date□□□□□□□□□□□□□□□□□□□□\_\_init\_\_()□□□□□□  
 □□□□□□□□□□□□□□□□□□□□□□□□

## 8.17 □□□□□init□□□□□

### 8.17.1 □□

□□□□□□□□□□□□□□□□□□□□□□□□\_\_init\_\_()□□□  
 □□□□□□□□□□

### 8.17.2 □□□□

□□□□□□□□\_\_new\_\_()□□□□□□□□□□□□□□□□□□  
 □□□□□□□□□□□□



```

class Date
:
    def
__init__(self, year, month, day):
    self.year = year
    self.month = month
    self.day = day

```

1. 在 Date 类中实现 \_\_init\_\_() 方法

```

>>> d = Date.__new__(Date)
>>> d
<__main__.Date object at 0x1006716d0>
>>> d.year
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'Date' object has no attribute 'year'
>>>

```

2. 遍历字典 data，将字典中的键值对赋值给 Date 对象的属性

```

>>> data = {'year':2012, 'month':8, 'day':29}
>>> for
key, value in
data.items():
...
    setattr(d, key, value)
...

```

```
>>> d.year
2012
>>> d.month
8
>>>
```

### 8.17.3 ☐☐

```

__init__()
deserializing
Date
today()

```

```
from time import

localtime

class Date:

    def

__init__(self, year, month, day):
    self.year = year
    self.month = month
    self.day = day

    @classmethod
    def

today(cls):
    d = cls.__new__(cls)
    t = localtime()
    d.year = t.tm_year
    d.month = t.tm_mon
    d.day = t.tm_mday
```

```
        return
    d
```

字典对象可以转换为JSON对象，方法如下：

```
data = { 'year': 2012, 'month': 8, 'day': 29 }
```

字典对象可以转换为Date对象，方法如下：

字典对象可以转换为字典对象，方法如下：

```
__dict__
__slots__
property
setattr()
```

## 8.18 Mixin

### 8.18.1

字典对象可以转换为字典对象，方法如下：

## 8.18.2 混入

Python 2.2 版本开始，Python 提供了混入（Mixin）的概念。混入是一种特殊的类，它不能被实例化，但可以被其他类继承。混入类通常用于实现一些通用的功能，比如日志记录、缓存等。

下面是一个简单的混入类 `mapping` 对象的例子，它实现了 `mapping` 对象的一些基本操作。这个混入类 `object` 是 `object` 的混入类。

```
class LoggedMappingMixin:
    """
    Add logging to get/set/delete operations for debugging.

    """
    __slots__ = ()

    def
    __getitem__(self, key):
        print
        ('Getting ' + str(key))
        return
        super().__getitem__(key)

    def
    __setitem__(self, key, value):
        print
        ('Setting {} = {!r}'.format(key, value))
        return
```

```

super().__setitem__(key, value)

    def
__delitem__(self, key):
    print
('Deleting ' + str(key))
    return

super().__delitem__(key)
class SetOnceMappingMixin
:
    ...
    Only allow a key to be set once.

    ...
    __slots__ = ()
    def
__setitem__(self, key, value):
    if

key in
self:
        raise KeyError
(str(key) + ' already set')
    return

super().__setitem__(key, value)
class StringKeysMappingMixin
:
    ...
    Restrict keys to strings only

    ...

```



```

del

d['x']
Deleting x

>>> from collections import

defaultdict
>>> class SetOnceDefaultDict

(SetOnceMappingMixin, defaultdict):
...     pass

...
>>>

d = SetOnceDefaultDict(list)
>>>

d['x'].append(2)
>>>

d['y'].append(3)
>>>

d['x'].append(10)
>>>

d['x'] = 23
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "mixin.py", line 24, in __setitem__
    raise KeyError

(str(key) + ' already set')
KeyError: 'x already set'

>>> from collections import

OrderedDict
>>> class StringOrderedDict

(StringKeysMappingMixin,

```

```

...             SetOnceMappingMixin,
...             OrderedDict):
...         pass

...
>>>

d = StringOrderedDict()
>>>

d['x'] = 23
>>>

d[42] = 10
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "mixin.py", line 45, in __setitem__
    ...
TypeError: keys must be strings
>>>

d['x'] = 42
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "mixin.py", line 46, in __setitem__
    __slots__ = ()
  File "mixin.py", line 24, in __setitem__
    if
key in self:
KeyError: 'x already set'
>>>

```

dict defaultdict OrderedDict

## 8.18.3



Python 的 mixin 类是 Python 中一个非常有趣的特性。它允许一个类从多个基类中继承方法，而不需要遵循严格的类层次结构。在 Python 中，mixin 类通常用于实现特定的功能，而不关心类层次结构。例如，socketserver 模块中的 ThreadingMixIn 就是一个典型的 mixin 类，它用于实现多线程的 XML-RPC 服务器。

```
from xmlrpc.server import
    SimpleXMLRPCServer
from socketserver import
    ThreadingMixIn
class ThreadedXMLRPCServer
    (ThreadingMixIn, SimpleXMLRPCServer):
        pass
```

在 Python 中，mixin 类通常用于实现特定的功能，而不关心类层次结构。例如，socketserver 模块中的 ThreadingMixIn 就是一个典型的 mixin 类，它用于实现多线程的 XML-RPC 服务器。

在 Python 中，mixin 类通常用于实现特定的功能，而不关心类层次结构。例如，socketserver 模块中的 ThreadingMixIn 就是一个典型的 mixin 类，它用于实现多线程的 XML-RPC 服务器。

在 Python 中，mixin 类通常用于实现特定的功能，而不关心类层次结构。例如，socketserver 模块中的 ThreadingMixIn 就是一个典型的 mixin 类，它用于实现多线程的 XML-RPC 服务器。



```

    isinstance(key, self.__restrict_key_type):
        raise TypeError

('Keys must be ' + str(self.__restrict_key_type))
    super().__setitem__(key, value)

```

□□□□□□□□□□□□□□□□

```

>>> class RDict
(RestrictKeysMixin, dict):
...     pass

...
>>>

d = RDict(_restrict_key_type=str)
>>>

e = RDict([('name','Dave'), ('n',37)],
_restrict_key_type=str)
>>>

f = RDict(name='Dave', n=37, _restrict_key_type=str)
>>>

f
{'n': 37, 'name': 'Dave'}
>>>

f[42] = 10
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "mixin.py", line 83, in __setitem__
    raise TypeError

('Keys must be ' + str(self.__restrict_key_type))
TypeError: Keys must be <class 'str'>
>>>

```

---

```
class RDict(dict):
    restrict_key_type = None
    mixin = None
```

```
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.mixin = self.mixin
        self.__getitem__ = self.__getitem__
        self.__setitem__ = self.__setitem__
        self.__delitem__ = self.__delitem__
        self.__iter__ = self.__iter__
        self.__len__ = self.__len__
        self.__repr__ = self.__repr__
        self.__str__ = self.__str__
        self.__bytes__ = self.__bytes__
        self.__hash__ = self.__hash__
        self.__eq__ = self.__eq__
        self.__neq__ = self.__neq__
        self.__lt__ = self.__lt__
        self.__le__ = self.__le__
        self.__gt__ = self.__gt__
        self.__ge__ = self.__ge__
        self.__contains__ = self.__contains__
        self.__getitem__ = self.__getitem__
        self.__setitem__ = self.__setitem__
        self.__delitem__ = self.__delitem__
        self.__iter__ = self.__iter__
        self.__len__ = self.__len__
        self.__repr__ = self.__repr__
        self.__str__ = self.__str__
        self.__bytes__ = self.__bytes__
        self.__hash__ = self.__hash__
        self.__eq__ = self.__eq__
        self.__neq__ = self.__neq__
        self.__lt__ = self.__lt__
        self.__le__ = self.__le__
        self.__gt__ = self.__gt__
        self.__ge__ = self.__ge__
        self.__contains__ = self.__contains__
```

```
class LoggedDict
(LoggedMappingMixin, dict):
    pass
```

```
class LoggedMappingMixin:
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.__getitem__ = self.__getitem__
        self.__setitem__ = self.__setitem__
        self.__delitem__ = self.__delitem__
        self.__iter__ = self.__iter__
        self.__len__ = self.__len__
        self.__repr__ = self.__repr__
        self.__str__ = self.__str__
        self.__bytes__ = self.__bytes__
        self.__hash__ = self.__hash__
        self.__eq__ = self.__eq__
        self.__neq__ = self.__neq__
        self.__lt__ = self.__lt__
        self.__le__ = self.__le__
        self.__gt__ = self.__gt__
        self.__ge__ = self.__ge__
        self.__contains__ = self.__contains__
```

## Mixin mixins mixins mixins mixins mixins mixins mixins

**def**

```
LoggedMapping(cls):  
    cls.__getitem__ = cls.__getitem__  
    cls.__setitem__ = cls.__setitem__  
    cls.__delitem__ = cls.__delitem__
```

**def**

```
__getitem__(self, key):  
    print
```

```
('Getting ' + str(key))  
    return
```

```
cls.__getitem__(self, key)
```

**def**

```
__setitem__(self, key, value):  
    print
```

```
('Setting {} = {!r}'.format(key, value))  
    return
```

```
cls.__setitem__(self, key, value)
```

**def**

```
__delitem__(self, key):  
    print
```

```
('Deleting ' + str(key))  
    return
```

```
cls.__delitem__(self, key)
```

```
cls.__getitem__ = __getitem__  
cls.__setitem__ = __setitem__  
cls.__delitem__ = __delitem__
```

```
    return  
  
    cls
```

□□□□□□□□□□□□□□□□□□□□□□□□

```
@LoggedMapping  
class LoggedDict  
  
    (dict):  
        pass
```

□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□9.12□□□□□

8.13□□□□□□□□□□□□□□□□mixin□□□□□□  
□□

## 8.19 □□□□□□□□□□□□□□

### 8.19.1 □□

□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□

## 8.19.2 连接池

连接池是一个容器，它包含了一些已经建立好的数据库连接。当应用程序需要连接数据库时，它可以从连接池中取出一个连接，使用完毕后，再将连接放回池中，以便其他应用程序使用。

```
class Connection
:
    def
__init__(self):
    self.state = 'CLOSED'

    def
read(self):
    if
self.state != 'OPEN':
        raise RuntimeError
('Not open')
    print
('reading')

    def
write(self, data):
    if
self.state != 'OPEN':
        raise RuntimeError
('Not open')
    print
('writing')

    def
```

```

open(self):
    if

self.state == 'OPEN':
    raise RuntimeError

('Already open')
    self.state = 'OPEN'

    def

close(self):
    if

self.state == 'CLOSED':
    raise RuntimeError

('Already closed')
    self.state = 'CLOSED'

```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
 □□□□□□□□□□□□□□□□□□□□□□□□□□□□  
 □read()□□□□write()□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
 Connection□□□□□□□□□□□□□□□□

```

class Connection
:
    def

__init__(self):
    self.new_state(ClosedConnectionState)

    def

```



```

new_state(self, newstate):
    self._state = newstate

    # Delegate to the state class

    def
read(self):
    return
self._state.read(self)

    def
write(self, data):
    return
self._state.write(self, data)

    def
open(self):
    return
self._state.open(self)

    def
close(self):
    return
self._state.close(self)

# Connection state base class

class ConnectionState
:
    @staticmethod
    def
read(conn):

```

```

        raise NotImplementedError

    ()

    @staticmethod
    def
write(conn, data):
    raise NotImplementedError

    ()

    @staticmethod
    def
open(conn):
    raise NotImplementedError

    ()

    @staticmethod
    def
close(conn):
    raise NotImplementedError

    ()

# Implementation of different states

class ClosedConnectionState

(ConnectionState):
    @staticmethod
    def

read(conn):
    raise RuntimeError

('Not open')

    @staticmethod
    def

```

```

write(conn, data):
    raise RuntimeError

('Not open')

    @staticmethod
    def

open(conn):
    conn.new_state(OpenConnectionState)

    @staticmethod
    def

close(conn):
    raise RuntimeError

('Already closed')

class OpenConnectionState

(ConnectionState):
    @staticmethod
    def

read(conn):
    print

('reading')

    @staticmethod
    def

write(conn, data):
    print

('writing')

    @staticmethod
    def

open(conn):
    raise RuntimeError

('Already open')

```

```

    @staticmethod
    def
close(conn):
    conn.new_state(ClosedConnectionState)

```

□□□□□□□□□□□□□□□□

```

>>> c = Connection()
>>> c._state
<class '__main__.ClosedConnectionState'>
>>> c.read()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "example.py", line 10, in read
    return
self._state.read(self)
  File "example.py", line 43, in read
    raise RuntimeError

('Not open')
RuntimeError: Not open
>>> c.open()
>>> c._state
<class '__main__.OpenConnectionState'>
>>> c.read()
reading
>>> c.write('hello')
writing
>>> c.close()
>>> c._state
<class '__main__.ClosedConnectionState'>
>>>

```

## 8.19.3 □□

```

    """
    """

```

```

    """
    """
    """Connection"""
    """
    """
    """Connection"""
    """
    """
    """NotImplementedError"""
    """8.12"""
    """

```

```

    """__class__"""
    """

```

```

class Connection
:
    def
__init__(self):
    self.new_state(ClosedConnection)

    def
new_state(self, newstate):
    self.__class__ = newstate

    def
read(self):
    raise NotImplementedError

```

```

()

    def
write(self, data):
    raise NotImplementedError

()

    def
open(self):
    raise NotImplementedError

()

    def
close(self):
    raise NotImplementedError

()

class ClosedConnection
(Connection):
    def
read(self):
    raise RuntimeError

('Not open')
    def
write(self, data):
    raise RuntimeError

('Not open')

    def
open(self):
    self.new_state(OpenConnection)

```

```

    def
close(self):
    raise RuntimeError
('Already closed')
class OpenConnection
(Connection):
    def
read(self):
    print
('reading')

    def
write(self, data):
    print
('writing')

    def
open(self):
    raise RuntimeError
('Already open')

    def
close(self):
    self.new_state(ClosedConnection)

```

Connection ConnectionState

```

>>> c = Connection()
>>> c
<__main__.ClosedConnection object at 0x1006718d0>
>>> c.read()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "state.py", line 15, in read
    raise RuntimeError

('Not open')
RuntimeError: Not open
>>> c.open()
>>> c
<__main__.OpenConnection object at 0x1006718d0>
>>> c.read()
reading
>>> c.close()
>>> c
<__main__.ClosedConnection object at 0x1006718d0>
>>>

```

```

class _
connection

```

```

if-elif-else

```

*# Original implementation*

```

class State
:
    def

```



```

__init__(self):
    self.state = 'A'
    def
action(self, x):
    if
state == 'A':
    # Action for A

    ...
    state = 'B'
    elif
state == 'B':
    # Action for B

    ...
    state = 'C'
    elif
state == 'C':
    # Action for C

    ...
    state = 'A'
# Alternative implementation

class State
:
    def
__init__(self):
    self.new_state(State_A)

    def
new_state(self, state):

```

```
        self.__class__ = state

    def
action(self, x):
    raise NotImplementedError

()

class State_A
(State):
    def
action(self, x):
    # Action for A

    ...
    self.new_state(State_B)

class State_B
(State):
    def
action(self, x):
    # Action for B

    ...
    self.new_state(State_C)

class State_C
(State):
    def
action(self, x):
    # Action for C

    ...
    self.new_state(State_A)
```

□Addison-Wesley, 1995□□□□□□□□□□□□□□□□  
□□

## 8.20.1 ☐☐

## 8.20.2

getattr()

```
import math

class Point:
    def
__init__(self, x, y):
    self.x = x
    self.y = y
```





## 8.21 □□□□□□□□

### 8.21.1 □□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

### 8.21.2 □□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
class Node
:
    pass

class UnaryOperator
(Node):
    def
__init__(self, operand):
    self.operand = operand

class BinaryOperator
(Node):
    def
```

```
__init__(self, left, right):  
    self.left = left  
    self.right = right
```

```
class Add
```

```
(BinaryOperator):  
    pass
```

```
class Sub
```

```
(BinaryOperator):  
    pass
```

```
class Mul
```

```
(BinaryOperator):  
    pass
```

```
class Div
```

```
(BinaryOperator):  
    pass
```

```
class Negate
```

```
(UnaryOperator):  
    pass
```

```
class Number
```

```
(Node):  
    def
```

```
__init__(self, value):
    self.value = value
```

[illegible]

```
# Representation of  $1 + 2 * (3 - 4) / 5$ 

t1 = Sub(Number(3), Number(4))
t2 = Mul(Number(2), t1)
t3 = Div(t2, Number(5))
t4 = Add(Number(1), t3)
```

```
t1 = Sub(Number(3), Number(4))
t2 = Mul(Number(2), t1)
t3 = Div(t2, Number(5))
t4 = Add(Number(1), t3)
```

[illegible]

```
class NodeVisitor
:
    def
visit(self, node):
    methname = 'visit_' + type(node).__name__
    meth = getattr(self, methname, None)
    if
meth is
None:
```



```

        meth = self.generic_visit
    return

meth(node)

    def
generic_visit(self, node):
    raise RuntimeError

('No {} method'.format('visit_' + type(node).__name__))

```

```

    visit_Name()
    Name

```

```

class Evaluator
(NodeVisitor):
    def
visit_Number(self, node):
    return
node.value

    def
visit_Add(self, node):
    return
self.visit(node.left) + self.visit(node.right)

    def
visit_Sub(self, node):
    return
self.visit(node.left) - self.visit(node.right)

```



```

generate_code(self, node):
    self.instructions = []
    self.visit(node)
    return

self.instructions

    def

visit_Number(self, node):
    self.instructions.append(('PUSH', node.value))

    def

binop(self, node, instruction):
    self.visit(node.left)
    self.visit(node.right)
    self.instructions.append((instruction,))

    def

visit_Add(self, node):
    self.binop(node, 'ADD')

    def

visit_Sub(self, node):
    self.binop(node, 'SUB')

    def

visit_Mul(self, node):
    self.binop(node, 'MUL')

    def

visit_Div(self, node):
    self.binop(node, 'DIV')

    def

unaryop(self, node, instruction):
    self.visit(node.operand)
    self.instructions.append((instruction,))

```

```
def visit_Negate(self, node):
    self.unaryop(node, 'NEG')
```

□ □ □ □ □ □ □ □ □ □ □ □ □

```
>>> s = StackCode()
>>> s.generate_code(t4)
[('PUSH', 1), ('PUSH', 2), ('PUSH', 3), ('PUSH', 4), ('SUB',),
 ('MUL',), ('PUSH', 5), ('DIV',), ('ADD',)]
>>>
```

### 8.21.3 ☐☐

```

    public void visit(Node node) {
        // ...
    }
}

```

```

    if

```

```
class NodeVisitor
:
    def
visit(self, node):
    nodetype = type(node). name
```



```
visit_Add(self, node):
    return

self.visit(node.left) + self.visit(node.right)
```

visit() Number

HTTP

```
class HTTPHandler
:
    def
handle(self, request):
    methname = 'do_' + request.request_method
    getattr(self, methname)(request)
    def
do_GET(self, request):
    ...
    def
do_POST(self, request):
    ...
    def
do_HEAD(self, request):
    ...
```

```

#####
#####Python#####
sys.getrecursionlimit()#####
#####Python #####
#####
#####
#####

```

8.22

```
Python ast 9.24 ast Python
```

**8.22** □□□□□□□□□□

### 8.22.1 □□

Python

## 8.22.2 ☐☐☐☐

## 8.21

```
import types

class Node
:
    pass

import types

class NodeVisitor
:
    def
visit(self, node):
    stack = [ node ]
    last_result = None
    while
stack:
        try
:
            last = stack[-1]
            if
isinstance(last, types.GeneratorType):
                stack.append(last.send(last_result))
                last_result = None
            elif
isinstance(last, Node):
```





```
class UnaryOperator
```

```
(Node):
```

```
    def
```

```
    __init__(self, operand):
```

```
        self.operand = operand
```

```
class BinaryOperator
```

```
(Node):
```

```
    def
```

```
    __init__(self, left, right):
```

```
        self.left = left
```

```
        self.right = right
```

```
class Add
```

```
(BinaryOperator):
```

```
    pass
```

```
class Sub
```

```
(BinaryOperator):
```

```
    pass
```

```
class Mul
```

```
(BinaryOperator):
```

```
    pass
```

```
class Div
```

```
(BinaryOperator):
```

```
    pass
```

```
class Negate
```

```
(UnaryOperator):  
    pass
```

```
class Number
```

```
(Node):  
    def
```

```
    __init__(self, value):  
        self.value = value
```

```
# A sample visitor class that evaluates expressions
```

```
class Evaluator
```

```
(NodeVisitor):  
    def
```

```
    visit_Number(self, node):  
        return
```

```
        node.value
```

```
        def
```

```
    visit_Add(self, node):  
        return
```

```
        self.visit(node.left) + self.visit(node.right)
```

```
        def
```

```
    visit_Sub(self, node):  
        return
```

```
        self.visit(node.left) - self.visit(node.right)
```

```
        def
```



```

>>> a = Number(0)
>>> for
n in
range(1, 100000):
...
    a = Add(a, Number(n))
...

>>> e = Evaluator()
>>> e.visit(a)
Traceback (most recent call last):
...

    File "visitor.py", line 29, in _visit
        return
meth(node)
    File "visitor.py", line 67, in visit_Add
        return
self.visit(node.left) + self.visit(node.right)
RuntimeError: maximum recursion depth exceeded
>>>

```

## □□□□□ Evaluator □□□□□□□□

```

class Evaluator
(NodeVisitor):
    def
visit_Number(self, node):
        return
node.value

```

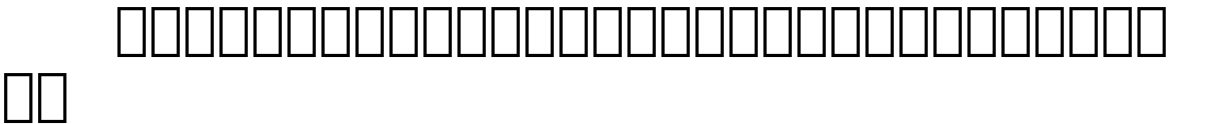
```
def
visit_Add(self, node):
    yield
    (yield
node.left) + (yield
node.right)

def
visit_Sub(self, node):
    yield
    (yield
node.left) - (yield
node.right)

def
visit_Mul(self, node):
    yield
    (yield
node.left) * (yield
node.right)

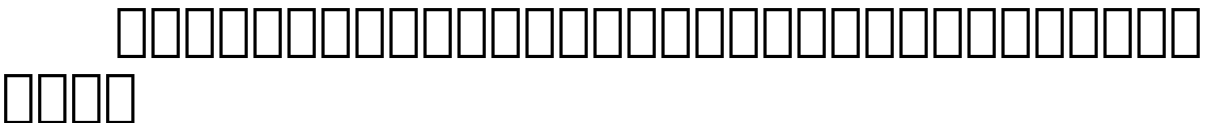
def
visit_Div(self, node):
    yield
    (yield
node.left) / (yield
node.right)
```

```
def
visit_Negate(self, node):
    yield
    -(yield
node.operand)
```



```
>>> a = Number(0)
>>> for
n in
range(1,100000):
...
    a = Add(a, Number(n))
...

>>> e = Evaluator()
>>> e.visit(a)
4999950000
>>>
```



```
class Evaluator
(NodeVisitor):
    ...
```







yield  
yield

return  
SyntaxError  
yield  
non-Node  
type  
last\_return  
last\_return  
yield

```
value = yield
node.left
```

value  
last\_return  
node.left

```
try
:
    last = stack[-1]
    if
isinstance(last, types.GeneratorType):
    stack.append(last.send(last_result))
    last_result = None
```



```

    last_result = 0
    return last_result

```

Node Node  
Node Node

```

class Visit
:
    def

__init__(self, node):
    self.node = node

class NodeVisitor
:
    def

visit(self, node):
    stack = [ Visit(node) ]
    last_result = None
    while

stack:
    try

:
        last = stack[-1]
        if

isinstance(last, types.GeneratorType):
            stack.append(last.send(last_result))
            last_result = None
        elif

```





## 8.23.1

## 8.23.2

```
import weakref

class Node:
    def __init__(self, value):
        self.value = value
        self._parent = None
        self.children = []

    def __repr__(self):
        return 'Node({!r})'.format(self.value)

    # property that manages the parent as a weak-reference

    @property
```

```
def
parent(self):
    return

self._parent if
self._parent is
None else

self._parent()

@parent.setter
def
parent(self, node):
    self._parent = weakref.ref(node)

def

add_child(self, child):
    self.children.append(child)
    child.parent = self
```

|   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |
|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|

```
>>> root = Node('parent')
>>> c1 = Node('child')
>>> root.add_child(c1)
>>> print

(c1.parent)
Node('parent')
>>> del

root
>>> print

(c1.parent)
None
```



```
>>>
```

## 8.23.3

Python

```
# Class just to illustrate when deletion occurs
```

```
class Data
```

```
:
```

```
    def
```

```
    __del__(self):  
        print
```

```
    ('Data.__del__')
```

```
# Node class involving a cycle
```

```
class Node
```

```
:
```

```
    def
```

```
    __init__(self):  
        self.data = Data()  
        self.parent = None  
        self.children = []
```

```
    def
```

```
    add_child(self, child):
```

```
self.children.append(child)
child.parent = self
```

A horizontal number line with 100 boxes, each representing 1%. The first two boxes are shaded blue, representing 2%.

```
>>> a = Data()
>>> del

a                                # Immediately deleted

Data.__del__
>>> a = Node()
>>> del

a                                # Immediately deleted

Data.__del__
>>> a = Node()
>>> a.add_child(Node())
>>> del

a                                # Not deleted (no message)

>>>
```

[illegible]





>>>

[illegible]

```
>>> a = Node()
>>> a_ref = weakref.ref(a)
>>> a_ref
<weakref at 0x100581f70; to 'Node' at 0x1005c5410>
>>>
```

```

    dereference
    None

```

```
(a_ref())
<__main__.Node object at 0x1005c5410>
>>> del
```

```
a
Data.__del__
>>> print
(a_ref())
None
>>>
```

$\frac{8.25}{0.75} = 11$

**8.24** □□□□□□□□

### 8.24.1 □□

[illegible]

## 8.24.2 ☐☐☐☐

```
Python
__ge__()

```

functools.total\_ordering  
\_\_eq\_\_()  
\_\_lt\_\_ \_\_le\_\_ \_\_gt\_\_ \_\_ge\_\_

```
from functools import
total_ordering
class Room
:
    def
__init__(self, name, length, width):
    self.name = name
    self.length = length
    self.width = width
    self.square_feet = self.length * self.width
@total_ordering
class House
:
    def
__init__(self, name, style):
    self.name = name
    self.style = style
    self.rooms = list()
    @property
    def
living_space_footage(self):
        return
```

```

sum(r.square_feet for
r in
self.rooms)

    def
add_room(self, room):
    self.rooms.append(room)

    def
__str__(self):
    return
'{}: {} square foot {}'.format(self.name,
self.living_space_footage,
                                self.style)

    def
__eq__(self, other):
    return
self.living_space_footage == other.living_space_footage

    def
__lt__(self, other):
    return
self.living_space_footage < other.living_space_footage

```

class House:
 @total\_ordering
 def \_\_eq\_\_(self, other):
 def \_\_lt\_\_(self, other):



```
# Build a few houses, and add rooms to them
```

```
h1 = House('h1', 'Cape')  
h1.add_room(Room('Master Bedroom', 14, 21))  
h1.add_room(Room('Living Room', 18, 20))  
h1.add_room(Room('Kitchen', 12, 16))  
h1.add_room(Room('Office', 12, 12))
```

```
h2 = House('h2', 'Ranch')  
h2.add_room(Room('Master Bedroom', 14, 21))  
h2.add_room(Room('Living Room', 18, 20))  
h2.add_room(Room('Kitchen', 12, 16))
```

```
h3 = House('h3', 'Split')  
h3.add_room(Room('Master Bedroom', 14, 21))  
h3.add_room(Room('Living Room', 18, 20))  
h3.add_room(Room('Office', 12, 16))  
h3.add_room(Room('Kitchen', 15, 17))  
houses = [h1, h2, h3]
```

```
print
```

```
('Is h1 bigger than h2?', h1 > h2) # prints True
```

```
print
```

```
('Is h2 smaller than h3?', h2 < h3) # prints True
```

```
print
```

```
('Is h2 greater than or equal to h1?', h2 >= h1) # Prints False
```

```
print
```

```
('Which one is biggest?', max(houses)) # Prints 'h3: 1101-square-foot Split'
```

```
print
```

```
('Which is smallest?', min(houses)) # Prints 'h2: 846-square-foot Ranch'
```

## 8.24.3

```
total_ordering(  
    # The following methods are needed for a class to be  
    # a total ordering.  
    __lt__(),  
    __le__(),  
    __gt__(),  
    __ge__()  
)
```

```
class House  
:  
    def  
__eq__(self, other):  
    ...  
    def  
__lt__(self, other):  
    ...  
    # Methods created by @total_ordering  
  
    __le__ = lambda  
self, other: self < other or  
self == other  
    __gt__ = lambda  
self, other: not
```

```

(self < other or
self == other)
    __ge__ = lambda
self, other: not
(self < other)
    __ne__ = lambda
self, other: not
self == other

```

@total\_ordering

## 8.25

### 8.25.1

logging

### 8.25.2

logging





## □□□□□□□□□□□□□□□□\_\_new\_\_()□□□

*# Note: This code doesn't quite work*

```
import weakref
```

```
class Spam
```

```
:
```

```
    _spam_cache = weakref.WeakValueDictionary()  
    def
```

```
__new__(cls, name):  
    if
```

```
name in
```

```
cls._spam_cache:  
    return
```

```
cls._spam_cache[name]  
    else
```

```
:
```

```
        self = super().__new__(cls)  
        cls._spam_cache[name] = self  
        return
```

```
self
```

```
    def
```

```
__init__(self, name):  
    print
```

```
('Initializing Spam')  
    self.name = name
```

\_\_init\_\_() 方法在创建新实例时调用。它接收一个字典，该字典包含实例的初始属性值。字典的键是属性名，值是属性值。字典可以是空的，也可以是包含多个键值对的字典。

```
>>> s = Spam('Dave')
Initializing Spam
>>> t = Spam('Dave')
Initializing Spam
>>> s is
t
True
>>>
```

WeakValueDictionary 字典的键是实例，值是任意对象。字典的键是实例，值是任意对象。字典的键是实例，值是任意对象。

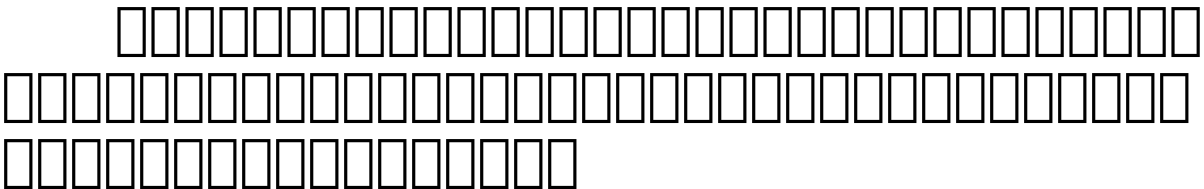
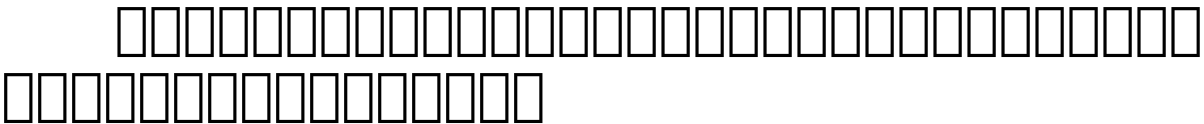
WeakValueDictionary 字典的键是实例，值是任意对象。字典的键是实例，值是任意对象。字典的键是实例，值是任意对象。

```
>>> a = get_spam('foo')
>>> b = get_spam('bar')
>>> c = get_spam('foo')
>>> list(_spam_cache)
['foo', 'bar']
>>> del
```

```
a
>>> del
c
```

```
>>> list(_spam_cache)
['bar']
>>> del

b
>>> list(_spam_cache)
[]
>>>
```



```
import weakref

class CachedSpamManager:
    def __init__(self):
        self._cache = weakref.WeakValueDictionary()

    def get_spam(self, name):
        if name not in self._cache:
            s = Spam(name)
            self._cache[name] = s
```





```
b
False
>>>
```

\_Spam

Spam\_\_\_\_\_init\_\_\_\_

```
:
    def
        __init__(self, *args, **kwargs):
            raise RuntimeError
                ("Can't instantiate directly")

                # Alternate constructor

        @classmethod
        def
            _new(cls, name):
                self = cls.__new__(cls)
                self.name = name
```

Spam.\_new() Spam() Spam()

```
import weakref

class CachedSpamManager:
    def __init__(self):
        self._cache = weakref.WeakValueDictionary()

    def get_spam(self, name):
        if name not in self._cache:
            s = Spam._new(name) # Modified creation
            self._cache[name] = s
        else:
            s = self._cache[name]
        return s
```

Spam Spam() Spam()

creational  
pattern9.13

---

[1] `obj == eval(repr(obj))`

[2]

[3] `Python__mro__`

[4] `super`

**9**

```

#####“#####Don't
repeat yourself”#####
#####
#####
Python#####“#####”#####
#####
#####Python#####
#####——#####exec()#####
#####——#####
#####Python
#####

```

## 9.1

### 9.1.1 □□

wrapper layer

## 9.1.2 口口口口

[illegible]

```

import time

from functools import
wraps
def
timethis(func):
    '''
        Decorator that reports the execution time.
    '''

    @wraps(func)
    def
wrapper(*args, **kwargs):
    start = time.time()
    result = func(*args, **kwargs)
    end = time.time()
    print
(func.__name__, end-start)
    return
result
    return
wrapper

```

□□□□□□□□□□□□□□

```

>>> @timethis
... def

```

```

countdown(n):
    ...
        ' ' '
    ...

    Counts down

    ...
        ' ' '
    ...

    while
n > 0:
    ...
        n -= 1
    ...

>>> countdown(100000)
countdown 0.008917808532714844
>>> countdown(10000000)
countdown 0.87188299392912
>>>

```

### 9.1.3

```
@timethis
def
countdown(n):
    ...
```

□□□□□□□□□□□□□□□□

```
def
countdown(n):
    ...
countdown = timethis(countdown)
```

□□□□□□□□□□□□□□ @staticmethod  
@classmethod□□@property□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□

```
class A
:
    @classmethod
    def
method(cls):
    pass

class B
:
    # Equivalent definition of a class method
```



```

def
method(cls):
    pass

method = classmethod(method)

```

```

def wrapper(*args,
**kwargs):
    """
    Wrapper function for the classmethod.
    It takes the same arguments as the
    original function and returns the
    same result.
    """
    return func(*args,
**kwargs)

```

```

def wrapper(*args,
**kwargs):
    """
    Wrapper function for the classmethod.
    It takes the same arguments as the
    original function and returns the
    same result.
    """
    return func(*args,
**kwargs)

```

```

@wraps(func)
def wrapper(*args,
**kwargs):
    """
    Wrapper function for the classmethod.
    It takes the same arguments as the
    original function and returns the
    same result.
    """
    return func(*args,
**kwargs)

```

## 9.2 装饰器函数

## 9.2.1

## 9.2.2

`functools`  
`@wraps`

```
import time

from functools import
wraps

def
timethis(func):
    '''
    Decorator that reports the execution time.

    '''

    @wraps(func)
    def
wrapper(*args, **kwargs):
    start = time.time()
    result = func(*args, **kwargs)
    end = time.time()
```



```
>>> countdown(100000)
countdown 0.008917808532714844
>>> countdown.__name__
'countdown'
>>> countdown.__doc__
'\n\tCounts down\n\t'
>>> countdown.__annotations__
{'n': <class 'int'>}
>>>
```

## 9.2.3 包装器

包装器函数（wrapper function）是一个接受一个或多个参数并返回一个函数的函数。它通常用于在不修改原函数的情况下，为函数添加新的功能。在 Python 中，装饰器（decorator）就是一种包装器函数，它通过 @wraps 函数来包装目标函数，以保持目标函数的元数据（如名称、文档字符串等）。

```
>>> countdown.__name__
'wrapper'
>>> countdown.__doc__
>>> countdown.__annotations__
{}
>>>
```

包装器函数通常具有以下属性：

- `@wraps`：用于包装目标函数，保持其元数据。
- `__wrapped__`：指向被包装的目标函数的属性。

```
>>> countdown.__wrapped__(100000)
>>>
```

`__wrapped__` 属性返回被包装函数的原始函数对象。

```
>>> from inspect import  
signature  
>>> print  
(signature(countdown))  
(n:int)  
>>>
```

在 Python 3.5 中，`inspect` 模块的 `signature` 函数可以返回一个函数对象的签名对象。这个对象包含函数的参数信息，包括参数名、参数类型、参数位置等。在 Python 3.5 之前，`inspect` 模块的 `signature` 函数返回的是一个字符串。在 Python 3.5 中，`inspect` 模块的 `signature` 函数返回的是一个 `Signature` 对象。这个对象包含函数的参数信息，包括参数名、参数类型、参数位置等。在 Python 3.5 之前，`inspect` 模块的 `signature` 函数返回的是一个字符串。在 Python 3.5 中，`inspect` 模块的 `signature` 函数返回的是一个 `Signature` 对象。这个对象包含函数的参数信息，包括参数名、参数类型、参数位置等。

## 9.3 装饰器

### 9.3.1 装饰器

装饰器是一种可以改变函数行为的工具。它通常是一个函数，接受一个函数作为参数，并返回一个新的函数对象。这个新的函数对象通常会对原始函数进行包装，并在调用原始函数之前或之后执行一些操作。装饰器通常用于日志记录、性能测量、认证授权、缓存等场景。

### 9.3.2 装饰器

Python 3.2 版本开始，`@wraps` 函数被引入，它用于在装饰器函数和被装饰函数之间复制元数据。在 Python 3.2 之前，装饰器函数需要手动复制元数据，这通常通过 `__wrapped__` 属性来实现。

```
>>> @somedecorator
>>> def
add(x, y):
...
    return
x + y
...

>>> orig_add = add.__wrapped__
>>> orig_add(3, 4)
7
>>>
```

### 9.3.3 装饰器

在 Python 3.2 之前，装饰器函数需要手动复制元数据。这通常通过 `__wrapped__` 属性来实现。在 Python 3.2 之后，`@wraps` 函数被引入，它用于在装饰器函数和被装饰函数之间复制元数据。在 Python 3.2 之前，装饰器函数需要手动复制元数据，这通常通过 `__wrapped__` 属性来实现。

在 Python 3.2 之前，装饰器函数需要手动复制元数据。这通常通过 `__wrapped__` 属性来实现。在 Python 3.2 之后，`@wraps` 函数被引入，它用于在装饰器函数和被装饰函数之间复制元数据。

# Python 3.3

```
from functools import
wraps

def
decorator1(func):
    @wraps(func)
    def
wrapper(*args, **kwargs):
    print

    ('Decorator 1')
    return

    func(*args, **kwargs)
    return

wrapper

def
decorator2(func):
    @wraps(func)
    def
wrapper(*args, **kwargs):
    print

    ('Decorator 2')
    return

    func(*args, **kwargs)
    return

wrapper

@decorator1
```

```
@decorator2
def
add(x, y):
    return
x + y
```

\_\_wrapped\_\_

```
>>> add(2, 3)
Decorator 1
Decorator 2
5
>>> add.__wrapped__(2, 3)
5
>>>
```

bug

<http://bugs.python.org/issue17482>

decorator chain

@wraps

@staticmethod @classmethod

descriptor

\_\_func\_\_



## 9.4 装饰器函数

### 9.4.1 简介

装饰器函数

### 9.4.2 装饰器

装饰器函数

```
from functools import
```

```
wraps
import logging
```

```
def
```

```
logged(level, name=None, message=None):
    '''
```

*Add logging to a function. level is the logging*

*level, name is the logger name, and message is the*

*log message. If name and message aren't specified,*

*they default to the function's module and name.*

```
'''

def
decorate(func):
    logname = name if
name else
func.__module__
    log = logging.getLogger(logname)
    logmsg = message if
message else
func.__name__

    @wraps(func)
    def
wrapper(*args, **kwargs):
    log.log(level, logmsg)
    return

func(*args, **kwargs)
    return

wrapper
    return

decorate

# Example use

@logged(logging.DEBUG)
def
add(x, y):
    return
```

```
x + y

@logged(logging.CRITICAL, 'example')
def

spam():
    print

('Spam!')
```

#####  
#####logged()#####  
#####decorate()#####  
#####logged()#####

## 9.4.3

#####  
#####

```
@decorator(x, y, z)
def

func(a, b):
    pass
```

#####

```
def
```

```
func = decorator(x, y, z)(func)
```

## 9.5

```

    accessor function nonlocal

```

```
from functools import
wraps, partial
import logging
```

```
# Utility decorator to attach a function as an attribute of  
obj
```

```
def
```

```
attach_wrapper(obj, func=None):
```

```
    if
```

```
func is
```

```
None:
```

```
    return
```

```
partial(attach_wrapper, obj)
```

```
    setattr(obj, func.__name__, func)
```

```
    return
```

```
func
```

```
def
```

```
logged(level, name=None, message=None):
```

```
    '''
```

```
        Add logging to a function. level is the logging
```

```
        level, name is the logger name, and message is the
```

```
        log message. If name and message aren't specified,
```

```
        they default to the function's module and name.
```

```
    '''
```

```
def
```

```

decorate(func):
    logname = name if
name else
func.__module__
    log = logging.getLogger(logname)
    logmsg = message if
message else
func.__name__
    @wraps(func)
    def
wrapper(*args, **kwargs):
    log.log(level, logmsg)
    return
func(*args, **kwargs)
    # Attach setter functions
    @attach_wrapper(wrapper)
    def
set_level(newlevel):
    nonlocal level
    level = newlevel
    @attach_wrapper(wrapper)
    def
set_message(newmsg):
    nonlocal logmsg
    logmsg = newmsg
    return
wrapper
return

```



```
>>> add.set_level(logging.WARNING)
>>> add(2, 3)
WARNING:__main__:Add called
5
>>>
```

## 9.5.3 装饰器

装饰器是Python中一种特殊的功能，用来将新的函数包裹在旧的函数中，从而对函数进行扩展，比如添加日志、性能统计、认证授权等。装饰器是Python中一种特殊的功能，用来将新的函数包裹在旧的函数中，从而对函数进行扩展，比如添加日志、性能统计、认证授权等。

装饰器是Python中一种特殊的功能，用来将新的函数包裹在旧的函数中，从而对函数进行扩展，比如添加日志、性能统计、认证授权等。装饰器是Python中一种特殊的功能，用来将新的函数包裹在旧的函数中，从而对函数进行扩展，比如添加日志、性能统计、认证授权等。

```
@timethis
@logged(logging.DEBUG)
def
countdown(n):
    while
n > 0:
    n -= 1
```

装饰器是Python中一种特殊的功能，用来将新的函数包裹在旧的函数中，从而对函数进行扩展，比如添加日志、性能统计、认证授权等。

```
>>> countdown(10000000)
DEBUG:__main__:countdown
countdown 0.8198461532592773
```



```
>>> countdown.set_level(logging.WARNING)
>>> countdown.set_message("Counting down to zero")
>>> countdown(10000000)
WARNING:__main__:Counting down to zero
countdown 0.8225970268249512
>>>
```

████████████████████████████████████████████████████████████████████████████████  
████████████████████

```
@logged(logging.DEBUG)
@timethis
def

countdown(n):
    while

n > 0:
    n -= 1
```

████████████████████████████████████████████████████████████████████████████████  
████████████████████████████████████████

```
...
@attach_wrapper(wrapper)
def

get_level():
    return

level

# Alternative

wrapper.get_level = lambda
```

```
: level
...
```

```
...
@wraps(func)
def
wrapper(*args, **kwargs):
    wrapper.log.log(wrapper.level, wrapper.logmsg)
    return

func(*args, **kwargs)

# Attach adjustable attributes

wrapper.level = level
wrapper.logmsg = logmsg
wrapper.log = log
...
```

`@timethis`

[illegible]

**9.6** □□□□□□□□□□□□□□□□

## 9.6.1 装饰器

装饰器（decorator）是 Python 中一个重要的概念，它提供了一种简洁、优雅的方式来修改或增强函数的功能。装饰器本质上是一个函数，它接受一个函数作为参数，并返回一个新的函数对象。这个新的函数对象通常会在原函数的基础上进行一些操作，比如记录函数的执行时间、添加日志、验证权限等。装饰器通过 @decorator 的语法糖来使用，使得代码更加简洁易读。在 Python 中，装饰器遵循一定的调用约定（calling convention），即装饰器函数应该接受一个函数对象作为参数，并返回一个函数对象。这种设计模式使得装饰器可以灵活地应用于各种场景，极大地提高了代码的可维护性和可扩展性。

## 9.6.2 装饰器

在 Python 9.5 版本中，装饰器被引入，为 Python 程序员提供了一种新的编程范式。装饰器允许我们以一种简单、直观的方式来修改函数的行为，而无需直接修改函数的源代码。这种设计模式不仅提高了代码的可读性，还使得代码更加模块化，易于维护和扩展。装饰器的使用极大地简化了诸如日志记录、性能监控、权限验证等任务的实现，使得开发者可以专注于业务逻辑的实现，而无需关心底层的装饰逻辑。

```
from functools import
wraps, partial
import logging

def
logged(func=None, *, level=logging.DEBUG, name=None,
message=None):
    if
func is
None:
        return
partial(logged, level=level, name=name, message=message)

    logname = name if
```

```
name else

func.__module__
    log = logging.getLogger(logname)
    logmsg = message if

message else

func.__name__
    @wraps(func)
    def

wrapper(*args, **kwargs):
    log.log(level, logmsg)
    return

func(*args, **kwargs)
    return

wrapper

# Example use

@logged
def

add(x, y):
    return

x + y

@logged(level=logging.CRITICAL, name='example')
def

spam():
    print

('Spam!')
```

```

    @logged
    @logged(level=logging.CRITICAL,
name='example')

```

### 9.6.3

programming consistency

```
@logged()
def
add(x, y):
    return
    x+y

```

```
# Example use

```

```
@logged
def
add(x, y):
    return
x + y
```

□□□□□□□□

```
def
add(x, y):
    return
x + y
add = logged(add)
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

logged□□□□□□□□□□logged()□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□

```
@logged(level=logging.CRITICAL, name='example')
def
spam():
    print
('Spam!')
```



typeassert은 type를 검사하는 decorator이다.  
@typeassert(int, int)를 사용하면, 함수의 인자가 int인지 검사한다.

```
>>> @typeassert(int, int)
... def
add(x, y):
...
    return
x + y
...

>>>
>>> add(2, 3)
5
>>> add(2, 'hello')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "contract.py", line 33, in wrapper
TypeError: Argument y must be <class 'int'>
>>>
```

@typeassert의 구현

```
from inspect import
signature
from functools import
wraps

def
typeassert(*ty_args, **ty_kwargs):
    def
```



```

decorate(func):
    # If in optimized mode, disable type checking

    if not
__debug__:
        return
func
    # Map function argument names to supplied types

    sig = signature(func)
    bound_types = sig.bind_partial(*ty_args,
**ty_kwargs).arguments

    @wraps(func)
    def
wrapper(*args, **kwargs):
    bound_values = sig.bind(*args, **kwargs)
    # Enforce type assertions across supplied
arguments

    for
name, value in
bound_values.arguments.items():
        if
name in
bound_types:
            if not
isinstance(value, bound_types[name]):
                raise TypeError
(
            'Argument {} must be {}'.format(name,

```





```

signature
>>> def

spam(x, y, z=42):
...

    pass

...

>>> sig = signature(spam)
>>> print

(sig)
(x, y, z=42)
>>> sig.parameters
mappingproxy(OrderedDict([('x', <Parameter at 0x10077a050
'x'>),
('y', <Parameter at 0x10077a158 'y'>), ('z', <Parameter at
0x10077a1b0 'z'>)]))
>>> sig.parameters['z'].name
'z'
>>> sig.parameters['z'].default
42
>>> sig.parameters['z'].kind
<_ParameterKind: 'POSITIONAL_OR_KEYWORD'>
>>>

```

bind\_partial()

```

>>> bound_types = sig.bind_partial(int,z=int)
>>> bound_types
<inspect.BoundArguments object at 0x10069bb50>
>>> bound_types.arguments
OrderedDict([('x', <class 'int'>), ('z', <class 'int'>)])
>>>

```

OrderedDict([('x', 1), ('y', 2), ('z', 3)])  
bound\_types.arguments['y']  
bound\_types.arguments['x']  
bound\_types.arguments['z']

sig.bind() bind()  
bind\_partial() partial()  
partial()

```
>>> bound_values = sig.bind(1, 2, 3)
>>> bound_values.arguments
OrderedDict([('x', 1), ('y', 2), ('z', 3)])
>>>
```

for name, value in bound\_values.arguments.items():

```
>>> for
name, value in
bound_values.arguments.items():
...
    if
name in
bound_types.arguments:
...
    if not
isinstance(value, bound_types.arguments[name]):
```

```

...
    raise TypeError
()
...

>>>

```

```

def bar(x, items=None):
    ...
    if
items is
None:
    ...
        items = []
    ...
        items.append(x)
    ...
    return
items
>>> bar(2)
[2]
>>> bar(2,3)
Traceback (most recent call last):

```

```
File "<stdin>", line 1, in <module>
  File "contract.py", line 33, in wrapper
TypeError: Argument items must be <class 'list'>
>>> bar(4, [1, 2, 3])
[1, 2, 3, 4]
>>>
```

function annotation

```
@typeassert
def
spam(x:int, y, z:int = 42):
    print
(x,y,z)
```

@typeassert

PEP 362  
<http://www.python.org/dev/peps/pep-0362>  
<http://docs.python.org/3/library/inspect.html>

## 9.8 □□□□□□□

### 9.8.1 □□

[illegible]

## 9.8.2 □□□□

```
from functools import
wraps

class A
:
    # Decorator as an instance method

    def
decorator1(self, func):
    @wraps(func)
    def
wrapper(*args, **kwargs):
        print
('Decorator 1')
        return
```



```

func(*args, **kwargs)
    return

wrapper

    # Decorator as a class method

    @classmethod
    def

decorator2(cls, func):
    @wraps(func)
    def

wrapper(*args, **kwargs):
    print

    ('Decorator 2')
    return

func(*args, **kwargs)
    return

wrapper

```

□□□□□□□□□□□□□□□□□□

```

# As an instance method

a = A()

@a.decorator1
def

spam():
    pass

```

```
@A.decorator2
def
grok():
    pass
```

### 9.8.3 ☐ ☐

```

    @property
    def getter():
    def setter():
    def deleter():

```

```
class Person
:
    # Create a property instance

    first_name = property()

    # Apply decorator methods

    @first_name.getter
    def
```

```

first_name(self):
    return

self._first_name

    @first_name.setter
    def

first_name(self, value):
    if not

isinstance(value, str):
    raise TypeError

('Expected a string')
    self._first_name = value

```

1. property 2. classmethod 3. staticmethod 4. class attribute 5. class method 6. class attribute 7. class method 8. class attribute 9. class method 10. class attribute

1. self 2. cls 3. decorator1() 4. decorator2() 5. self 6. cls 7. decorator1() 8. decorator2() 9. self 10. cls 11. decorator1() 12. decorator2() 13. self 14. cls 15. decorator1() 16. decorator2() 17. self 18. cls 19. decorator1() 20. decorator2()

1. A 2. B 3. A 4. B 5. A 6. B 7. A 8. B 9. A 10. B 11. A 12. B 13. A 14. B 15. A 16. B 17. A 18. B 19. A 20. B

```

class B

(A):
    @A.decorator2
    def

bar(self):
    pass

```

1. 在类 B 中定义一个方法 bar，并为其添加装饰器 @A.decorator2。

## 9.9 装饰器

### 9.9.1 装饰器

装饰器是一种可以改变函数行为的函数。通常，装饰器接受一个函数作为参数，并返回一个新的函数对象。

### 9.9.2 装饰器

装饰器通常通过修改函数的 \_\_call\_\_ 方法来实现。\_\_call\_\_ 方法是 Python 中所有对象的内置方法，用于调用对象。

```

import types

from functools import
wraps

class Profiled
:
    def

__init__(self, func):
    wraps(func)(self)
    self.ncalls = 0

    def

__call__(self, *args, **kwargs):
    self.ncalls += 1
    return

self.__wrapped__(*args, **kwargs)

    def

__get__(self, instance, cls):
    if

instance is

None:
        return

self

    else

:
        return

types.MethodType(self, instance)

```

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  
XXX

```
@Profiled
def
add(x, y):
    return
x + y
class Spam
:
    @Profiled
    def
bar(self, x):
    print
(self, x)
```

XXXXXXXXXXXXXXXXXXXXXXXXXXXX

```
>>> add(2, 3)
5
>>> add(4, 5)
9
>>> add.ncalls
2
>>> s = Spam()
>>> s.bar(1)
<__main__.Spam object at 0x10069e9d0> 1
>>> s.bar(2)
<__main__.Spam object at 0x10069e9d0> 2
>>> s.bar(3)
<__main__.Spam object at 0x10069e9d0> 3
>>> Spam.bar.ncalls
```

## 9.9.3 装饰器

装饰器（decorator）是 Python 中一种特殊的功能，用来装饰（即包裹）一个函数，以改变其功能。装饰器通常是一个高阶函数，它接受一个函数作为参数，并返回一个新的函数对象。这个新的函数对象通常会在原函数的基础上进行一些修改，比如添加日志、性能统计、权限检查等。

Python 的 `functools` 模块提供了 `wraps()` 函数，用于装饰器函数。它的主要作用是复制被装饰函数的元数据（如文档字符串、名称等）到装饰后的函数上，从而保持函数的可追溯性。

在 Python 中，装饰器通常通过 `@decorator` 的语法来使用。例如，`@wraps(func)` 就是一个常见的装饰器。它会将 `func` 的元数据复制到装饰后的函数上。此外，装饰器也可以直接作为函数调用来使用，如 `decorator(func)`。

```
>>> s = Spam()
>>> s.bar(3)
Traceback (most recent call last):
...
TypeError: spam() missing 1 required positional argument: 'x'
```

在 Python 3 中，装饰器函数通常使用 `@decorator` 的语法。例如，`@wraps(func)` 是一个装饰器，它会将 `func` 的元数据复制到装饰后的函数上。此外，装饰器也可以直接作为函数调用来使用，如 `decorator(func)`。

```

>>> s = Spam()
>>> def
grok(self, x):
...
    pass
...

>>> grok.__get__(s, Spam)
<bound method Spam.grok of <__main__.Spam object at
0x100671e90>>
>>>

```

```

    __get__()
type. MethodType()
__get__()instanceNone
Profiledncalls

```

```

9.5
nonlocal

```

```

import types

from functools import
wraps

def

```



```

profiled(func):
    ncalls = 0
    @wraps(func)
    def

wrapper(*args, **kwargs):
    nonlocal ncalls
    ncalls += 1
    return

func(*args, **kwargs)
    wrapper.ncalls = lambda

: ncalls
    return

wrapper

# Example

@profiled
def

add(x, y):
    return

x + y

```

```

ncalls

```

```

>>> add(2, 3)
5
>>> add(4, 5)
9
>>> add.ncalls()
2
>>>

```

## 9.10 装饰器函数

### 9.10.1 简介

装饰器函数

### 9.10.2 装饰器

装饰器函数  
@classmethod  
@staticmethod  
@property

```
import time

from functools import
wraps

# A simple decorator

def
timethis(func):
    @wraps(func)
    def
wrapper(*args, **kwargs):
    start = time.time()
    r = func(*args, **kwargs)
    end = time.time()
    print
```

```

(end-start)
    return

r
    return

wrapper

# Class illustrating application of the decorator to different
kinds of methods

class Spam

:
    @timethis
    def

instance_method(self, n):
    print

    (self, n)
    while

n > 0:
        n -= 1

        @classmethod
        @timethis
        def

class_method(cls, n):
    print

    (cls, n)
    while

n > 0:
        n -= 1

        @staticmethod
        @timethis
        def

static_method(n):

```

```

    print
(n)
while
n > 0:
    n -= 1

```

```
>>> s = Spam()
>>> s.instance_method(1000000)
<__main__.Spam object at 0x1006a6050> 1000000
0.11817407608032227
>>> Spam.class_method(1000000)
<class '__main__.Spam'> 1000000
0.11334395408630371
>>> Spam.static_method(1000000)
1000000
0.11740279197692871
>>>
```

### 9.10.3 ☐☐

```
class Spam
:
    ...
    @timethis
    @staticmethod
```

```

def
static_method(n):
    print
(n)
    while
n > 0:
    n -= 1

```

static\_method

```

>>> Spam.static_method(1000000)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "timethis.py", line 6, in wrapper
    start = time.time()
TypeError: 'staticmethod' object is not callable
>>>

```

@classmethod  
 @staticmethod  
 8.9  
 @classmethod  
 @staticmethod

8.12

```

from abc import
ABCMeta, abstractmethod

class A

    (metaclass=ABCMeta):
        @classmethod
        @abstractmethod
        def

method(cls):
    pass

```

@classmethod  
 @abstractmethod  
 def

## 9.11

### 9.11.1

@classmethod  
 @abstractmethod  
 def

### 9.11.2

keyword-only arguments

```
from functools import
wraps

def
optional_debug(func):
    @wraps(func)
    def
wrapper(*args, debug=False, **kwargs):
    if
debug:
        print
('Calling', func.__name__)
    return
func(*args, **kwargs)
    return
wrapper
```

keyword-only arguments

```
>>> @optional_debug
... def
spam(a,b,c):
...
    print
(a,b,c)
```

```
...

>>> spam(1,2,3)
1 2 3
>>> spam(1,2,3, debug=True)
Calling spam
1 2 3
>>>
```

## 9.11.3 递归

```

def a(x, debug=False):
    if debug:
        print('Calling a')
    ...
def b(x, y, z, debug=False):
    if debug:
        print('Calling b')
    ...

```



```
def
c(x, y, debug=False):
    if
debug:
        print
('Calling c')
    ...
```

□□□□□□□□□□□□□□□□

```
@optional_debug
def
a(x):
    ...

@optional_debug
def
b(x, y, z):
    ...

@optional_debug
def
c(x, y):
    ...
```

□□□□□□□□□□□□□□□□ keyword-only □□  
 □□□□□□□□□□□□\*args□\*\*kwargs□□□□□□□□□□  
 keyword-only□□□□□□□□□□□□□□□□□□□□□□□□  
 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□



wrapper

```
>>> @optional_debug
... def

add(x,y):
...

    return

x+y
...

>>> import inspect

>>> print

(inspect.signature(add))
(x, y)
>>>
```

[illegible]

```
from functools import  
wraps  
import inspect  
  
def
```





```

def
log_getattribute(cls):
    # Get the original implementation

    orig_getattribute = cls.__getattribute__

    # Make a new definition

    def
new_getattribute(self, name):
    print
('getting:', name)
    return
orig_getattribute(self, name)

    # Attach to the class and return

    cls.__getattribute__ = new_getattribute
    return
cls
# Example use

@log_getattribute
class A
:
    def
__init__(self,x):
    self.x = x
    def
spam(self):
    pass

```

□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> a = A(42)
>>> a.x
getting: x
42
>>> a.spam()
getting: spam
>>>
```

### 9.12.3 □□

□□□□□□□□□□□□□□□□□□□□**mixin**□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□

```
class LoggedGetattribute
:
    def
__getattr__(self, name):
    print
('getting:', name)
    return
super().__getattr__(name)
# Example:
```

```
class A
    (LoggedGetattribute):
        def
            __init__(self,x):
                self.x = x
            def
                spam(self):
                    pass
```

```

class MRO:
    def __init__(self, cls):
        self.cls = cls
        self.mro = self._get_mro()

    def _get_mro(self):
        mro = []
        cls = self.cls
        while cls:
            mro.append(cls)
            cls = cls.__base__
        return mro

    def __str__(self):
        return f'MRO of {self.cls.__name__}: {self.mro}'

    def __repr__(self):
        return f'MRO({self.cls.__name__})'

# Example usage
class A:
    pass

class B(A):
    pass

class C(B):
    pass

# Create an instance of MRO
mro = MRO(C)

# Print the MRO
print(mro)  # Output: MRO of C: [C, B, A, object]
print(mro.__str__())  # Output: MRO of C: [C, B, A, object]
print(mro.__repr__())  # Output: MRO(C)

```

8.13

## 9.13



## 9.13.1 ☐☐

## 9.13.2 ☐☐☐☐

# Python

```
class Spam
:
    def
__init__(self, name):
    self.name = name

a = Spam('Guido')
b = Spam('Diana')
```

`__call__()`

```
class NoInstances
    (type):
        def
            call (self, *args, **kwargs):
```



```
class Singleton
```

```
(type):
```

```
    def
```

```
    __init__(self, *args, **kwargs):
```

```
        self.__instance = None
```

```
        super().__init__(*args, **kwargs)
```

```
        def
```

```
        __call__(self, *args, **kwargs):
```

```
            if
```

```
self.__instance is
```

```
None:
```

```
                self.__instance = super().__call__(*args,  
**kwargs)
```

```
            return
```

```
self.__instance
```

```
        else
```

```
        :
```

```
            return
```

```
self.__instance
```

```
# Example
```

```
class Spam
```

```
(metaclass=Singleton):
```

```
    def
```

```
    __init__(self):
```

```
        print
```

```
('Creating Spam')
```

□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> a = Spam()  
Creating Spam  
>>> b = Spam()  
>>> a is  
  
b  
True  
>>> c = Spam()  
>>> a is  
  
c  
True  
>>>
```

□□□□□□□□□□□□□□□□cached instance□  
8.25□□□□□□□□□□□□□□□□□□□□

```
import weakref  
  
class Cached  
(type):  
    def  
  
    __init__(self, *args, **kwargs):  
        super().__init__(*args, **kwargs)  
        self.__cache = weakref.WeakValueDictionary()  
  
    def  
  
    __call__(self, *args):  
        if  
  
args in
```

```

self.__cache:
    return

self.__cache[args]
    else

:
    obj = super().__call__(*args)
    self.__cache[args] = obj
    return

obj

# Example

class Spam

(metaclass=Cached):
    def

__init__(self, name):
    print

('Creating Spam({!r})'.format(name))
    self.name = name

```

□□□□□□□□□□□□□□□□

```

>>> a = Spam('Guido')
Creating Spam('Guido')
>>> b = Spam('Diana')
Creating Spam('Diana')
>>> c = Spam('Guido')           # Cached

>>> a is
b
False
>>> a is

```

### 9.13.3

```
class _Spam
:
    def
__init__(self):
        print
('Creating Spam')
_spam_instance = None
def
Spam():
    global
_spam_instance
    if
_spam_instance is not
None:
        return
```

```
_spam_instance
else
:
    _spam_instance = _Spam()
    return
_spam_instance
```

字典的键可以是任意对象，但必须是可哈希的。字典的值可以是任意对象。

8.25 字典的键可以是任意对象，但必须是可哈希的。字典的值可以是任意对象。字典的键可以是任意对象，但必须是可哈希的。字典的值可以是任意对象。

## 9.14 字典的键可以是任意对象，但必须是可哈希的。

### 9.14.1 字典的键可以是任意对象，但必须是可哈希的。

字典的键可以是任意对象，但必须是可哈希的。字典的值可以是任意对象。字典的键可以是任意对象，但必须是可哈希的。字典的值可以是任意对象。

### 9.14.2 字典的键可以是任意对象，但必须是可哈希的。

字典的键可以是任意对象，但必须是可哈希的。字典的值可以是任意对象。字典的键可以是任意对象，但必须是可哈希的。字典的值可以是任意对象。

```
from collections import
```

```
OrderedDict
```

```
# A set of descriptors for various types
```

```
class Typed
```

```
:
```

```
    _expected_type = type(None)  
    def
```

```
__init__(self, name=None):  
    self._name = name
```

```
    def
```

```
__set__(self, instance, value):  
    if not
```

```
isinstance(value, self._expected_type):  
        raise TypeError
```

```
    ('Expected ' + str(self._expected_type))  
        instance.__dict__[self._name] = value
```

```
class Integer
```

```
(Typed):  
    _expected_type = int
```

```
class Float
```

```
(Typed):  
    _expected_type = float
```

```
class String
```

```
(Typed):  
    _expected_type = str
```

```
# Metaclass that uses an OrderedDict for class body
```

```
class OrderedMeta
```



```

(type):
    def

__new__(cls, clsname, bases, clsdict):
    d = dict(clsdict)
    order = []
    for
name, value in
clsdict.items():
    if
isinstance(value, Typed):
        value._name = name
        order.append(name)
    d['_order'] = order
    return

type.__new__(cls, clsname, bases, d)
@classmethod
def

__prepare__(cls, clsname, bases):
    return

OrderedDict()

```

OrderedDict

\_order\_order
CSV

```

class Structure
(metaclass=OrderedMeta):

```



## 9.14.3

`__prepare__()` 方法由 `OrderedMeta` 类实现，用于在创建 `mapping object` 时，将 `OrderedDict` 的初始化参数（如 `clsname`）传递给 `__init__` 方法。

在 `__init__` 方法中，我们首先检查 `name` 是否已经存在于 `self` 中。如果存在，则抛出 `TypeError` 异常。

```
from collections import
OrderedDict

class NoDupOrderedDict
(OrderedDict):
    def
    __init__(self, clsname):
        self.clsname = clsname
        super().__init__()
    def
    __setitem__(self, name, value):
        if
        name in
        self:
            raise TypeError
        ('{} already defined in {}'.format(name, self.clsname))
```



```
pass
```

```
...
```

```
def
```

```
spam(self):
```

```
...
```

```
pass
```

```
...
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
File "<stdin>", line 4, in A
```

```
File "dupmethod2.py", line 25, in __setitem__  
(name, self.clsname))
```

```
TypeError: spam already defined in A
```

```
>>>
```

```
class __new__():  
    def __init__(self):  
        self.__dict__ = dict()  
        d = dict(clsdict)
```

```
class ORM:  
    def __init__(self):  
        self.__dict__ = dict()
```

```
class Stock
```

```
(Model):  
    name = String()  
    shares = Integer()  
    price = Float()
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

## 9.15 □□□□□□□□□□□□□□□□

### 9.15.1 □□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

### 9.15.2 □□□□

□□□□□□□□Python□□□□□class□□□□□□□□  
metaclass□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□

```
from abc import  
ABCMeta, abstractmethod
```

```
class IStream

(metaclass=ABCMeta):
    @abstractmethod
    def

read(self, maxsize=None):
    pass

    @abstractmethod
    def

write(self, data):
    pass
```

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

```
class Spam
(metaclass=MyMeta, debug=True, synchronize=True):
    ...
```

```

__prepare__().__new__().__init__()
keyword-only

```

```
class MyMeta
```

```
(type):
    # Optional

    @classmethod
    def

__prepare__(cls, name, bases, *, debug=False,
synchronize=False):
    # Custom processing

    ...
    return
super().__prepare__(name, bases)

    # Required

    def

__new__(cls, name, bases, ns, *, debug=False,
synchronize=False):
    # Custom processing

    ...
    return
super().__new__(cls, name, bases, ns)

    # Required

    def

__init__(self, name, bases, ns, *, debug=False,
synchronize=False):
    # Custom processing

    ...
    super().__init__(name, bases, ns)
```



## 9.15.3 类

类是Python中一种特殊的对象，它封装了数据（属性）和函数（方法）。类是创建对象的模板，通过类可以创建出任意数量的对象。类通常包含以下方法：

- `__prepare__()`：在类创建时调用，用于准备类字典。
- `__new__()`：在类创建时调用，用于创建类对象。
- `__init__()`：在类创建时调用，用于初始化类对象。

类对象通常包含以下属性：

- `__new__()`：用于创建类对象的方法。
- `__init__()`：用于初始化类对象的方法。
- `__prepare__()`：用于准备类字典的方法。
- `__prepare__()`：用于准备类字典的方法。

类对象通常包含以下属性：

- `keyword-only`：用于指定关键字参数。

类对象通常包含以下属性：

- ...

```
class Spam
```

```
(metaclass=MyMeta):
    debug = True
    synchronize = True
```



```

>>> from inspect import
Signature, Parameter
>>> # Make a signature for a func(x, y=42, *, z=None)

>>> parms = [ Parameter('x', Parameter.POSITIONAL_OR_KEYWORD),
...
                Parameter('y', Parameter.POSITIONAL_OR_KEYWORD,
default=42),
...
                Parameter('z', Parameter.KEYWORD_ONLY,
default=None) ]
>>> sig = Signature(parms)
>>> print

(sig)
(x, y=42, *, z=None)
>>>

```

bind()
 \*args \*\*kwargs

```

>>> def
func(*args, **kwargs):
...
    bound_values = sig.bind(*args, **kwargs)
...
    for
name, value in
bound_values.arguments.items():
...

```

```

    print
(name,value)
...

>>> # Try various examples

>>> func(1, 2, z=3)
x 1
y 2
z 3
>>> func(1)
x 1
>>> func(1, z=3)
x 1
z 3
>>> func(y=2, x=1)
x 1
y 2
>>> func(1, 2, 3, 4)
Traceback (most recent call last):
...

File "/usr/local/lib/python3.3/inspect.py", line 1972, in
_bind
    raise TypeError
('too many positional arguments')
TypeError: too many positional arguments
>>> func(y=2)
Traceback (most recent call last):
...

File "/usr/local/lib/python3.3/inspect.py", line 1961, in
_bind
    raise TypeError
(msg) from None

TypeError: 'x' parameter lacking default value

```



```

:
    __signature__ = make_sig()
def
__init__(self, *args, **kwargs):
    bound_values = self.__signature__.bind(*args,
**kwargs)
    for
name, value in
bound_values.arguments.items():
    setattr(self, name, value)

# Example use

class Stock
(Structure):
    __signature__ = make_sig('name', 'shares', 'price')

class Point
(Structure):
    __signature__ = make_sig('x', 'y')

```

□□□□□□□□□□Stock□□□□□□□□

```

>>> import inspect

>>> print
(inspect.signature(Stock))
(name, shares, price)
>>> s1 = Stock('ACME', 100, 490.1)
>>> s2 = Stock('ACME', 100)
Traceback (most recent call last):
...

```



```
Signature(parms)
```

```
class StructureMeta
```

```
(type):  
    def
```

```
    __new__(cls, clsname, bases, clsdict):  
        clsdict['__signature__'] =  
        make_sig(*clsdict.get('_fields', []))  
        return
```

```
    super().__new__(cls, clsname, bases, clsdict)
```

```
class Structure
```

```
(metaclass=StructureMeta):  
    _fields = []  
    def
```

```
    __init__(self, *args, **kwargs):  
        bound_values = self.__signature__.bind(*args,  
        **kwargs)  
        for
```

```
        name, value in
```

```
        bound_values.arguments.items():  
            setattr(self, name, value)
```

```
# Example
```

```
class Stock
```

```
(Structure):  
    _fields = ['name', 'shares', 'price']
```

```
class Point
```

```
(Structure):  
    _fields = ['x', 'y']
```



inspect 模块提供了对 Python 对象 introspection 的能力。  
\_\_signature\_\_ 属性提供了对函数或方法的签名。  
inspect 模块还提供了 introspection 模块。  
inspect 模块提供了对 Python 对象 introspection 的能力。

```
>>> import inspect

>>> print
(inspect.signature(Stock))
(name, shares, price)
>>> print
(inspect.signature(Point))
(x, y)
>>>
```

## 9.17 模块 introspection

### 9.17.1 模块

模块 introspection 模块提供了对 Python 模块 introspection 的能力。  
inspect 模块还提供了 introspection 模块。

### 9.17.2 模块

模块 introspection 模块提供了对 Python 模块 introspection 的能力。  
inspect 模块还提供了 introspection 模块。  
type 模块提供了对 Python 模块 introspection 的能力。  
\_\_new\_\_() 方法。

`__init__()`□□□□□□□□□□

```
class MyMeta
(type):
    def
__new__(self, clsname, bases, clsdict):
    # clsname is name of class being defined

    # bases is tuple of base classes

    # clsdict is class dictionary

    return
super().__new__(cls, clsname, bases, clsdict)
```

□□□□□□□□□□`__init__()`□

```
class MyMeta
(type):
    def
__init__(self, clsname, bases, clsdict):
    super().__init__(clsname, bases, clsdict)
    # clsname is name of class being defined

    # bases is tuple of base classes

    # clsdict is class dictionary
```





```
class B
```

(Root) :

def

```
fooBar(self):
```

```
# TypeError
```

**pass**

[illegible]

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

```
from inspect import
```

signature

```
import logging
```

```
class MatchSignaturesMeta
```

```
(type):
```

def

```
__init__(self, clsname, bases, clsdict):
```

```
super().__init__(clsname, bases, clsdict)
```

```
sup = super(self, self)
```

for

```
name, value in
```

```
clsdict.items():
```

**if**

```
name.startswith('_') or not
```

```

callable(value):
    continue

    # Get the previous definition (if any) and compare
    the signatures

    prev_dfn = getattr(sup,name,None)
    if

prev_dfn:
    prev_sig = signature(prev_dfn)
    val_sig = signature(value)
    if

prev_sig != val_sig:
    logging.warning('Signature mismatch in %s.
%s != %s',
                    value.__qualname__, prev_sig,
val_sig)
# Example

class Root
    (metaclass=MatchSignaturesMeta):
        pass

class A
    (Root):
        def

foo(self, x, y):
    pass

def

spam(self, x, *, z):

```

```
pass
```

```
# Class with redefined methods, but slightly different signatures
```

```
class B
```

```
(A):
```

```
def
```

```
foo(self, a, b):
```

```
pass
```

```
def
```

```
spam(self,x,z):
```

```
pass
```

```
□□□□□□□□□□□□□□□□
```

```
WARNING:root:Signature mismatch in B.spam. (self, x, *, z) !=  
(self, x, z)  
WARNING:root:Signature mismatch in B.foo. (self, x, y) !=  
(self, a, b)
```

```
□□□□□□□□□□□□□□□□bug□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□
```

## 9.17.3 工厂函数

工厂函数（factory function）是指返回一个对象的函数。在Python中，工厂函数通常用于创建对象，而不是返回一个值。工厂函数的名称通常以“factory”结尾。

工厂函数的主要优点是它们可以返回不同类型的对象，而不仅仅是返回一个值。这使得工厂函数非常灵活，可以用于创建各种类型的对象。工厂函数的另一个优点是它们可以封装对象的创建逻辑，使得代码更加简洁和易于维护。

工厂函数通常使用`__new__()`和`__init__()`方法来创建对象。`__new__()`方法用于创建对象，而`__init__()`方法用于初始化对象。工厂函数通常返回一个已经初始化好的对象，而不是返回一个未初始化的对象。工厂函数的名称通常以“factory”结尾。

工厂函数在Python中非常常见，尤其是在创建对象时。工厂函数的名称通常以“factory”结尾。工厂函数的主要优点是它们可以返回不同类型的对象，而不仅仅是返回一个值。这使得工厂函数非常灵活，可以用于创建各种类型的对象。工厂函数的另一个优点是它们可以封装对象的创建逻辑，使得代码更加简洁和易于维护。



```
        super(self, self).__init__(
            self.__dict__
        )
        self.__dict__
```

## 9.18 9.18.1

### 9.18.1

```
        exec(
            """
            """
        )
```

### 9.18.2

```
        types.new_class(
            "class dictionary",
            {}
        )
```

```
# stock.py
```

```
# Example of making a class manually from parts
```

```
# Methods
```

```
def
__init__(self, name, shares, price):
    self.name = name
    self.shares = shares
    self.price = price

def
cost(self):
    return
self.shares * self.price

cls_dict = {
    '__init__' : __init__,
    'cost' : cost,
}

# Make a class

import types

Stock = types.new_class('Stock', (), {}, lambda
ns: ns.update(cls_dict))
Stock.__module__ = __name__
```

[illegible]

```
>>> s = Stock('ACME', 50, 91.1)
>>> s
<stock.Stock object at 0x1006a9b10>
>>> s.cost()
4555.0
```

```
>>>
```

```
types.new_class()
Stock.__module__
__module__
__repr__()
pickle
__module__
```

```
types.new_class()
```

```
>>> import abc
```

```
>>> Stock = types.new_class('Stock', (), {'metaclass':
abc.ABCMeta},
```

```
...                               lambda
```

```
ns: ns.update(cls_dict))
```

```
...
```

```
>>> Stock.__module__ = __name__
```

```
>>> Stock
```

```
<class
```

```
'__main__
```

```
.Stock'>
```

```
>>> type(Stock)
```

```
<class
```

```
'abc
```

```
.ABCMeta'>
```

```
>>>
```

```
class Spam
(Base, debug=True, typecheck=False):
    ...
```

```
new_class()
```

```
Spam = types.new_class('Spam', (Base,),
                        {'debug': True, 'typecheck': False},
                        lambda
ns: ns.update(cls_dict))
```

```
new_class()
__prepare__()
9.14
update()
```

### 9.18.3 ☐☐

collections.namedtuple() 创建命名元组

```
>>> Stock = collections.namedtuple('Stock', ['name', 'shares', 'price'])
>>> Stock
<class '__main__.Stock'>
>>>
```

namedtuple() 与 exec() 一起使用

```
import operator

import types

import sys

def
named_tuple(classname, fieldnames):
    # Populate a dictionary of field property accessors

    cls_dict = { name: property(operator.itemgetter(n))
for
n, name in
enumerate(fieldnames) }

    # Make a __new__ function and add to the class dict
```

```

def
__new__(cls, *args):
    if
len(args) != len(fieldnames):
        raise TypeError
('Expected {} arguments'.format(len(fieldnames)))
    return
tuple.__new__(cls, args)
    cls_dict['__new__'] = __new__
    # Make the class

    cls = types.new_class(classname, (tuple,), {},
        lambda
ns: ns.update(cls_dict))

    # Set the module to that of the caller

    cls.__module__ = sys._getframe(1).f_globals['__name__']
    return
cls

```

“frame hack”
 sys.\_getframe()
 frame
 hack
 2.15

2.15

```

>>> Point = namedtuple('Point', ['x', 'y'])
>>> Point
<class '__main__.Point'>
>>> p = Point(4, 5)
>>> len(p)
2
>>> p.x
4
>>> p.y
5
>>> p.x = 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: can't set attribute
>>> print

('%s %s' % p)
4 5
>>>

```

```

class Stock:
    def __init__(self, price):
        self.price = price
    def __str__(self):
        return 'Stock(%s)' % self.price

```

```

Stock = type('Stock', (), cls_dict)

```

```

class Stock:
    def __init__(self, price):
        self.price = price
    def __str__(self):
        return 'Stock(%s)' % self.price
    def __prepare__(self, name, globals):
        return types.new_class(name, (object,), {'__init__': self.__init__, '__str__': self.__str__})

```

types.prepare\_class()

```
import types
```

```
metaclass, kwargs, ns = types.prepare_class('Stock', (),  
{'metaclass': type})
```

types.prepare\_class()

PEP 3115  
<http://www.python.org/dev/peps/pep-3115> Python  
<http://docs.python.org/3/reference/datamodel.html#metaclasses>

## 9.19

### 9.19.1



## 9.19.2 元组

元组是不可变的有序集合。它们通常用于存储一组相关的值。元组可以包含任何类型的对象，包括其他元组。元组的创建和访问与列表类似，但元组一旦创建就不能修改。

元组与集合（collections）不同，集合是无序且不重复的。元组是有序的，并且可以包含重复的元素。

```
import operator

class StructTupleMeta(type):
    def __init__(cls, *args, **kwargs):
        super().__init__(*args, **kwargs)
        for n, name in enumerate(cls._fields):
            setattr(cls, name,
                    property(operator.itemgetter(n)))

class StructTuple(tuple, metaclass=StructTupleMeta):
    _fields = []
    def __new__(cls, *args):
        if len(args) != len(cls._fields):
            raise ValueError('{} arguments
required'.format(len(cls._fields)))
        return super().__new__(cls, args)
```

元组的创建和访问示例：

```
class Stock(StructTuple):
    _fields = ['name', 'shares', 'price']

class Point(StructTuple):
    _fields = ['x', 'y']
```

```
>>> s = Stock('ACME', 50, 91.1)
>>> s
('ACME', 50, 91.1)
>>> s[0]
'ACME'
>>> s.name
'ACME'
>>> s.shares * s.price
4555.0
>>> s.shares = 23
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: can't set attribute
>>>
```

```

    StructTupleMeta(fields)
    """
    operator.itemgetter()
    """
    accessor function
    """
    property()
    """
    property
    """

```

```

StructTupleMeta.__init__()
cls = ...
fields = ...

```

StructTuple 클래스를 만들 때 \_\_new\_\_() 메서드를 구현할 때 \_\_new\_\_() 메서드를 구현할 때 \_\_new\_\_() 메서드를 구현할 때

```
s = Stock('ACME', 50, 91.1)          # OK

s = Stock(('ACME', 50, 91.1))        # Error
```

\_\_init\_\_() 메서드와 \_\_new\_\_() 메서드를 구현할 때 immutable 클래스를 만들 때 \_\_init\_\_() 메서드를 구현할 때 \_\_new\_\_() 메서드를 구현할 때

Python 3.3 버전에서 \_\_new\_\_() 메서드를 구현할 때

## PEP 422

<http://www.python.org/dev/peps/pep-0422> PEP 422는 Python 3.3 버전에서 \_\_new\_\_() 메서드를 구현할 때

## 9.20 多态性

### 9.20.1 多态性

多态性是指一个对象可以具有多种不同的形态。在 Python 中，多态性可以通过多种不同的方式实现，其中最常见的方式是通过使用 `multiple-dispatch` 库来实现。

### 9.20.2 多态性

多态性是指一个对象可以具有多种不同的形态。在 Python 中，多态性可以通过多种不同的方式实现，其中最常见的方式是通过使用 `multiple-dispatch` 库来实现。

```
class Spam:
    def
bar(self, x:int, y:int):
    print
('Bar 1:', x, y)
    def
bar(self, s:str, n:int = 0):
    print
('Bar 2:', s, n)

s = Spam()
s.bar(2, 3)           # Prints Bar 1: 2 3
```

```
s.bar('hello')           # Prints Bar 2: hello 0
```

[illegible]

```
# multiple.py
```

```
import inspect
```

```
import types
```

```
class MultiMethod
```

■ ■ ■

/// /// ///

*Represents a single multimethod.*

*///*

def

```
__init__(self, name):
    self._methods = {}
    self.__name__ = name
```

def

```
register(self, meth):
    '''
```

```

    Register a new method as a multimethod

    '''

    sig = inspect.signature(meth)

    # Build a type signature from the method's annotations

    types = []
    for
name, parm in
sig.parameters.items():
        if
name == 'self':
            continue

        if
parm.annotation is
inspect.Parameter.empty:
            raise TypeError

(
            'Argument {} must be annotated with a
type'.format(name)
        )
        if not
isinstance(parm.annotation, type):
            raise TypeError

(
            'Argument {} annotation must be a
type'.format(name)

```

```

        )
        if
parm.default is not
inspect.Parameter.empty:
            self._methods[tuple(types)] = meth
            types.append(parm.annotation)

        self._methods[tuple(types)] = meth

    def
__call__(self, *args):
    '''
        Call a method based on type signature of the arguments
    '''

    types = tuple(type(arg) for
arg in
args[1:])
    meth = self._methods.get(types, None)
    if
meth:
        return
meth(*args)
    else
:
        raise TypeError
('No matching method for types {}'.format(types))

    def
__get__(self, instance, cls):

```

```

        '''

        Descriptor method needed to make calls work in a class

        '''

        if
instance is not
None:
            return
types.MethodType(self, instance)
        else
:
            return
self
class MultiDict
(dict):
    '''

    Special dictionary to build multimethods in a metaclass

    '''

    def
__setitem__(self, key, value):
        if
key in
self:
            # If key already exists, it must be a multimethod

```



*or callable*

```
        current_value = self[key]
        if
isinstance(current_value, MultiMethod):
            current_value.register(value)
        else
:
            mvalue = MultiMethod(key)
            mvalue.register(current_value)
            mvalue.register(value)
            super().__setitem__(key, mvalue)
        else
:
            super().__setitem__(key, value)

class MultipleMeta
(type):
    '''
        Metaclass that allows multiple dispatch of methods
        ...

    def
__new__(cls, clsname, bases, clsdict):
    return
type.__new__(cls, clsname, bases, dict(clsdict))

    @classmethod
    def
__prepare__(cls, clsname, bases):
    return
```

```
MultiDict()
```

```
□□□□□□□□□□□□□□□□
```

```
class Spam
    (metaclass=MultipleMeta):
        def
bar(self, x:int, y:int):
    print
    ('Bar 1:', x, y)
    def
bar(self, s:str, n:int = 0):
    print
    ('Bar 2:', s, n)
# Example: overloaded __init__

import time

class Date
    (metaclass=MultipleMeta):
        def
__init__(self, year: int, month:int, day:int):
    self.year = year
    self.month = month
    self.day = day

    def
__init__(self):
    t = time.localtime()
```

```
self.__init__(t.tm_year, t.tm_mon, t.tm_mday)
```

□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> s = Spam()
>>> s.bar(2, 3)
Bar 1: 2 3
>>> s.bar('hello')
Bar 2: hello 0
>>> s.bar('hello', 5)
Bar 2: hello 5
>>> s.bar(2, 'hello')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "multiple.py", line 42, in __call__
    raise TypeError

('No matching method for types {}'.format(types))
TypeError: No matching method for types (<class 'int'>, <class
'str'>)

>>> # Overloaded __init__

>>> d = Date(2012, 12, 21)
>>> # Get today's date

>>> e = Date()
>>> e.year
2012
>>> e.month
12
>>> e.day
3
>>>
```

## 9.20.3 □□



```
>>> b.__self__
<__main__.Spam object at 0x1006a46d0>
>>> b.__func__
<__main__.MultiMethod object at 0x1006a4d50>
>>> b(2, 3)
Bar 1: 2 3
>>> b('hello')
Bar 2: hello 0
>>>
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> s.bar(x=2, y=3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: __call__() got an unexpected keyword argument 'y'

>>> s.bar(s='hello')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: __call__() got an unexpected keyword argument 's'
>>>
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□\_\_call\_\_()□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□

```
class A
```

```
:
```

```
    pass
```

```
class B
```

```
(A):
```

```
    pass
```

```
class C
```

```
:
```

```
    pass
```

```
class Spam
```

```
(metaclass=MultipleMeta):
```

```
    def
```

```
foo(self, x:A):
```

```
    print
```

```
('Foo 1:', x)
```

```
    def
```

```
foo(self, x:C):
```

```
    print
```

```
('Foo 2:', x)
```

#####x:A#####B#####  
#####

```
>>> s = Spam()
>>> a = A()
>>> s.foo(a)
Foo 1: <__main__.A object at 0x1006a5310>
>>> c = C()
>>> s.foo(c)
Foo 2: <__main__.C object at 0x1007a1910>
>>> b = B()
>>> s.foo(b)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "multiple.py", line 44, in __call__
    raise TypeError

('No matching method for types {}'.format(types))
TypeError: No matching method for types (<class
'__main__.B'>,)
>>>
```

#####  
#####

```
import types

class multimethod
:
    def

__init__(self, func):
    self._methods = {}
    self.__name__ = func.__name__
```

```

        self._default = func

    def
match(self, *types):
    def
register(func):
        ndefaults = len(func.__defaults__) if
func.__defaults__ else
0
        for
n in
range(ndefaults+1):
            self._methods[types[:len(types) - n]] = func
            return
self
    return
register
    def
__call__(self, *args):
        types = tuple(type(arg) for
arg in
args[1:])
        meth = self._methods.get(types, None)
        if
meth:
            return
meth(*args)
        else
:
            return

```



```

self._default(*args)

    def
__get__(self, instance, cls):
    if
instance is not
None:
        return
types.MethodType(self, instance)
    else
:
        return
self

```

□□□□□□□□□□□□□□□□□□□□□□

```

class Spam
:
    @multimethod
    def
bar(self, *args):
    # Default method called if no match

    raise TypeError
('No matching method for bar')

    @bar.match(int, int)
    def

```



## 9.21 面向对象编程

### 9.21.1 类

面向对象编程是一种编程范式，它将数据（对象）和行为（方法）封装在一起。在Python中，类（Class）是创建对象的蓝图，对象是类的实例。类定义了对象的属性和方法，而对象则是这些属性和方法的具体实现。

### 9.21.2 类属性

类属性（Class Attribute）是定义在类中的属性，它们属于类本身，而不是类的实例。类属性通常用于定义常量或共享数据。在Python中，类属性可以通过@property装饰器来定义，这允许我们以属性访问的方式调用类方法。

```
class Person:
    def
__init__(self, name ,age):
    self.name = name
    self.age = age

    @property
    def
name(self):
        return
self._name

    @name.setter
    def
name(self, value):
        if not
```



```

    @property
    def
prop(self):
    return

getattr(self, storage_name)

    @prop.setter
    def
prop(self, value):
    if not

isinstance(value, expected_type):
    raise TypeError

('{} must be a {}'.format(name, expected_type))
    setattr(self, storage_name, value)
    return

prop

# Example use

class Person

:
    name = typed_property('name', str)
    age = typed_property('age', int)
    def

__init__(self, name, age):
    self.name = name
    self.age = age

```

## 9.21.3 □□

————  
typed\_property()  
typed\_property()  
getter setter  
name expected\_type storage\_name  
————

functools.partial()

```
from functools import
partial

String = partial(typed_property, expected_type=str)
Integer = partial(typed_property, expected_type=int)
# Example:

class Person
:
    name = String('name')
    age = Integer('age')
    def
__init__(self, name, age):
    self.name = name
    self.age = age
```

8.13

## 9.22

### 9.22.1

with

### 9.22.2

contextlib@contextmanager

```
import time

from contextlib import
contextmanager

@contextmanager
def
timethis(label):
    start = time.time()
    try
:
        yield
```





```
orig_list[:] = working
```

```
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
```

```
>>> items = [1, 2, 3]

>>> with list_transaction(items) as working:

...
    working.append(4)

...
    working.append(5)

...

>>> items

[1, 2, 3, 4, 5]

>>> with list_transaction(items) as working:

...
    working.append(6)
```

```
...
    working.append(7)

...
    raise RuntimeError('oops')

...

Traceback (most recent call last):

  File "<stdin>", line 4, in <module>

RuntimeError: oops

>>> items

[1, 2, 3, 4, 5]

>>>
```

### 9.22.3 ☐☐

```

__enter__()__exit__()

```

```

import time

class timethis:

    def __init__(self, label):

        self.label = label

    def __enter__(self):

        self.start = time.time()

    def __exit__(self, exc_ty, exc_val, exc_tb):

        end = time.time()

        print('{}: {}'.format(self.label, end - self.start))

```

```

    @contextmanager

```

```

    @contextmanager
    def self-
        contained
        with
        __enter__()
        __exit__()

```

## 9.23 子プロセスの作成

### 9.23.1 実行

`exec()` は、現在のプロセスで、与えられたコマンドを実行する。実行されたコマンドは、現在のプロセスの環境変数、グローバル変数、ローカル変数、および関数に影響を与える。

### 9.23.2 例

以下は、`exec()` を使用して、現在のプロセスで、与えられたコマンドを実行する例である。

```
>>> a = 13

>>> exec('b = a + 1')

>>> print(b)

14

>>>
```

この例では、`a` が 13 に設定され、`exec()` が実行された後、`b` が 14 になる。

```
>>> def test():
```

```
...
```

```
    a=13
```

```
...
```

```
    exec('b = a + 1')
```

```
...
```

```
    print(b)
```

```
...
```

```
>>> test()
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
  File "<stdin>", line 4, in test
```

```
NameError: global name 'b' is not defined
```

```
>>>
```

raise NameError('exec() can only be used in the global scope')

locals() can only be used in the global scope

```
>>> def
test():
...
    a = 13
...
    loc = locals()
...
    exec
('b = a + 1')
...
    b = loc['b']
...    print
(b)
...
```

```
>>> test()
14
>>>
```

## 9.23.3

exec() 函数可以执行一个字符串表示的任意 Python 语句。  
exec() 函数语法如下：  
exec(statement, globals, locals)

exec() 函数有三个参数，第一个参数是要执行的 Python 语句，第二个参数是全局作用域（字典），第三个参数是局部作用域（字典）。  
exec() 函数可以执行一个字符串表示的任意 Python 语句。  
exec() 函数可以执行一个字符串表示的任意 Python 语句。  
exec() 函数可以执行一个字符串表示的任意 Python 语句。  
exec() 函数可以执行一个字符串表示的任意 Python 语句。  
exec() 函数可以执行一个字符串表示的任意 Python 语句。  
exec() 函数可以执行一个字符串表示的任意 Python 语句。

```
>>> def
test1():
...
    x = 0
...
    exec
('x += 1')
...
    print
(x)
...

>>> test1()
0
>>>
```

locals() exec() exec()

```
>>> def
test2():
...
    x = 0
...
    loc = locals()
...
    print
('before:', loc)
...
    exec
('x += 1')
...
    print
('after:', loc)
...
    print
('x =', x)
...

>>> test2()
before: {'x': 0}
after: {'loc': {...}, 'x': 1}
x = 0
>>>
```



```
loc[x] = 0
x = 1
```

```
locals()
locals()
locals()
```

```
>>> def
test3():
...
    x = 0
...
    loc = locals()
...
    print
(loc)
...
    exec
('x += 1')
...
    print
(loc)
...
    locals()
...
```

```
    print
(loc)
...

>>> test3()
{'x': 0}
{'loc': {...}, 'x': 1}
{'loc': {...}, 'x': 0}
>>>
```

□□□□□locals()□□□□□□□□x□□□□□

□□□□□locals()□□□□□□□□□□□□□□□□□□□□□□  
exec()□□□□□□

```
>>> def
test4():
...
    a = 13
...
    loc = { 'a' : a }
...
    glb = { }
...
    exec
('b = a + 1', glb, loc)
...
    b = loc['b']
...

```

```
print
(b)
...

>>> test4()
14
>>>
```

exec()은 실행할 코드를 문자열로 받거나, 코드가 있는 파일을 파일명으로 받습니다. exec()은 실행할 코드를 문자열로 받거나, 코드가 있는 파일을 파일명으로 받습니다.

exec()은 실행할 코드를 문자열로 받거나, 코드가 있는 파일을 파일명으로 받습니다. exec()은 실행할 코드를 문자열로 받거나, 코드가 있는 파일을 파일명으로 받습니다.

## 9.24 Python

### 9.24.1

Python은 실행할 코드를 문자열로 받거나, 코드가 있는 파일을 파일명으로 받습니다.

### 9.24.2

Python은 실행할 코드를 문자열로 받거나, 코드가 있는 파일을 파일명으로 받습니다.

```

>>> x = 42
>>> eval('2 + 3*4 + x')
56
>>> exec

('for i in range(10): print(i)')
0
1
2
3
4
5
6
7
8
9
>>>

```

Pythonのastモジュール  
 PythonのAST (Abstract Syntax Tree)

```

>>> import ast

>>> ex = ast.parse('2 + 3*4 + x', mode='eval')
>>> ex
<_ast.Expression object at 0x1007473d0>
>>> ast.dump(ex)
"Expression(body=BinOp(left=BinOp(left=Num(n=2), op=Add(),
right=BinOp(left=Num(n=3), op=Mult(), right=Num(n=4))),
op=Add(),
right=Name(id='x', ctx=Load())))"

>>> top = ast.parse('for i in range(10): print(i)',
mode='exec')
>>> top
<_ast.Module object at 0x100747390>
>>> ast.dump(top)
"Module(body=[For(target=Name(id='i', ctx=Store()),
iter=Call(func=Name(id='range', ctx=Load()), args=[Num(n=10)]),

```

```

keywords=[], starargs=None, kwargs=None),
body=[Expr(value=Call(func=Name(id='print', ctx=Load()),
args=[Name(id='i', ctx=Load())], keywords=[], starargs=None,
kwargs=None))], orelse=[])]"
>>>

```

AST
 visit\_NodeName()
 NodeName

```

import ast

class CodeAnalyzer(ast.NodeVisitor):
    def __init__(self):
        self.loaded = set()
        self.stored = set()
        self.deleted = set()
    def visit_Name(self, node):
        if isinstance(node.ctx, ast.Load):
            self.loaded.add(node.id)
        elif isinstance(node.ctx, ast.Store):
            self.stored.add(node.id)
        elif isinstance(node.ctx, ast.Del):
            self.deleted.add(node.id)

# Sample usage
if __name__ == '__main__':
    # Some Python code
    code = '''
for i in range(10):
    print(i)
del i
'''

    # Parse into an AST
    top = ast.parse(code, mode='exec')

```

```
# Feed the AST to analyze name usage
c = CodeAnalyzer()
c.visit(top)
print('Loaded:', c.loaded)
print('Stored:', c.stored)
print('Deleted:', c.deleted)
```

□□□□□□□□□□□□□□□□

```
Loaded: {'i', 'range', 'print'}
Stored: {'i'}
Deleted: {'i'}
```

□□□AST□□□□□□compile()□□□□□□□□□□□□  
□□□

```
>>> exec(compile(top, '<stdin>', 'exec'))
```

0

1

2

3

4

5

6

```
7
```

```
8
```

```
9
```

```
>>>
```

## 9.24.3 练习

编写一个名为 `namelower` 的函数，它接受一个 AST 对象作为参数，并返回一个字典，其中键是全局访问的变量名，值是它们的名称大小写。该函数应该使用 `exec()` 来执行 AST 对象中的代码，并返回一个字典，其中键是全局访问的变量名，值是它们的名称大小写。

编写一个名为 `namelower` 的函数，它接受一个 AST 对象作为参数，并返回一个字典，其中键是全局访问的变量名，值是它们的名称大小写。该函数应该使用 `exec()` 来执行 AST 对象中的代码，并返回一个字典，其中键是全局访问的变量名，值是它们的名称大小写。

```
# namelower.py
import ast
import inspect

# Node visitor that lowers globally accessed names into
```

```

# the function body as local variables.
class NameLower(ast.NodeVisitor):
    def __init__(self, lowered_names):
        self.lowered_names = lowered_names

    def visit_FunctionDef(self, node):
        # Compile some assignments to lower the constants
        code = '__globals = globals()\n'
        code += '\n'.join("{0} = "
__globals['{0}']".format(name)
                                for name in self.lowered_names)
        code_ast = ast.parse(code, mode='exec')

        # Inject new statements into the function body
        node.body[:0] = code_ast.body

        # Save the function object
        self.func = node

# Decorator that turns global names into locals
def lower_names(*namelist):
    def lower(func):
        srclines = inspect.getsource(func).splitlines()
        # Skip source lines prior to the @lower_names
decorator
        for n, line in enumerate(srclines):
            if '@lower_names' in line:
                break

        src = '\n'.join(srclines[n+1:])
        # Hack to deal with indented code
        if src.startswith((' ', '\t')):
            src = 'if 1:\n' + src
        top = ast.parse(src, mode='exec')

        # Transform the AST
        cl = NameLower(namelist)
        cl.visit(top)

        # Execute the modified AST
        temp = {}
        exec(compile(top, '', 'exec'), temp, temp)

        # Pull out the modified code object
        func.__code__ = temp[func.__name__].__code__

```





[compile-time/](#) Python  
AST

## 9.25 Python

### 9.25.1

Python

### 9.25.2

dis Python

```
>>> def countdown(n):
...     while n > 0:
...         print('T-minus', n)
...         n -= 1
...         print('Blastoff!')
...
>>> import dis
>>> dis.dis(countdown)
2          0 SETUP_LOOP                39 (to 42)
           >>    3 LOAD_FAST              0 (n)
               6 LOAD_CONST             1 (0)
               9 COMPARE_OP             4 (>)
              12 POP_JUMP_IF_FALSE      41

3          15 LOAD_GLOBAL              0 (print)
              18 LOAD_CONST             2 ('T-minus')
              21 LOAD_FAST              0 (n)
              24 CALL_FUNCTION         2 (2 positional, 0
```

```
keyword pair)
    27 POP_TOP

    4      28 LOAD_FAST          0 (n)
           31 LOAD_CONST        3 (1)
           34 INPLACE_SUBTRACT
           35 STORE_FAST        0 (n)
           38 JUMP_ABSOLUTE
           >> 41 POP_BLOCK

    5      >> 42 LOAD_GLOBAL      0 (print)
           45 LOAD_CONST        4 ('Blastoff!')
           48 CALL_FUNCTION      1 (1 positional, 0
keyword pair)
           51 POP_TOP          0 (None)
           52 LOAD_CONST
           55 RETURN_VALUE

>>>
```

### 9.25.3 ☐ ☐

dis

dis()

```
>>> countdown.__code__.co_code
b"x'\x00|\x00\x00d\x01\x00k\x04\x00r)\x00t\x00\x00d\x02\x00|\x00\x00\x83
\x02\x00\x01|\x00\x00d\x03\x008}\x00\x00q\x03\x00wt\x00\x00d\x04\x00\x83
\x01\x00\x01d\x00\x00S"
>>>
```

opcode

```
>>> c = countdown.__code__.co_code
>>> import opcode

>>> opcode.opname[c[0]]
>>> opcode.opname[c[0]]
'SETUP_LOOP'
>>> opcode.opname[c[3]]
'LOAD_FAST'
>>>
```

dis

```
import opcode

def
generate_opcodes(codebytes):
    extended_arg = 0
    i = 0
    n = len(codebytes)
    while
i < n:
    op = codebytes[i]
    i += 1
    if
op >= opcode.HAVE_ARGUMENT:
```

```

        oparg = codebytes[i] + codebytes[i+1]*256 +
extended_arg
        extended_arg = 0
        i += 2
        if

op == opcode.EXTENDED_ARG:
        extended_arg = oparg * 65536
        continue

    else

:
        oparg = None
    yield
(op, oparg)

```

□□□□□□□□□□□□□□□□

```

>>> for
op, oparg in
generate_opcodes(countdown.__code__.co_code):
...
    print
(op, opcode.opname[op], oparg)
...

120 SETUP_LOOP 39
124 LOAD_FAST 0
100 LOAD_CONST 1
107 COMPARE_OP 4
114 POP_JUMP_IF_FALSE 41
116 LOAD_GLOBAL 0
100 LOAD_CONST 2

```





## 10 命名空间

Python 解释器在启动时会创建一个全局命名空间，这个命名空间是一个字典，它包含了所有的全局变量。在 Python 中，命名空间（namespace）是用来组织代码的一种方式，它允许我们在同一个程序中定义具有相同名称的变量，而不会发生冲突。命名空间的概念在 Python 中非常重要，因为它决定了变量在程序中的可见性和作用域。在 Python 中，命名空间的管理是通过 `__name__`、`__package__` 和 `__import__` 等属性来实现的。

### 10.1 命名空间的类型

#### 10.1.1 全局命名空间

全局命名空间是 Python 解释器在启动时创建的一个命名空间，它包含了所有的全局变量。

#### 10.1.2 局部命名空间

局部命名空间是在函数调用时创建的，它包含了函数内部定义的变量。局部命名空间的名称通常以 `__init__.py` 文件的形式存在，用于初始化模块。

```
graphics/  
  __init__.py  
  primitive/  
    __init__.py  
    line.py  
    fill.py  
    text.py
```



```
formats/  
    __init__.py  
    png.py  
    jpg.py
```

```
from typing import *
```

```
import graphics.primitive.line

from graphics.primitive import
line
import graphics.formats.jpg as jpg
```

### 10.1.3 练习

```

import graphics
from graphics import *
import graphics.formats.jpg
from graphic import *
graphics/formats/__init__.py
graphics/formats/jpg.py

```



`__init__.py` □□□□□□□□

## 10.2

## 10.2.1 ☐☐

```
from module import *

```

## 10.2.2 ☐☐☐☐

[illegible]

```
# somemodule.py
```

def

```
spam() :  
    pass
```

def

```
grok():
    pass
```

```
blah = 42

# Only export 'spam' and 'grok'

__all__ = ['spam', 'grok']
```

## 10.2.3 `__all__`

When you use `from module import *`, Python searches for the `__all__` attribute in the module. If it finds it, it uses the list of names in `__all__` to determine which names to import. If it doesn't find it, it imports all public names (names not starting with an underscore).

If you have a module with a `__all__` attribute, and you use `import module`, Python will raise an `AttributeError` if you try to access a name that is not in `__all__`.

## 10.3 `__init__.py`

### 10.3.1 `__init__.py`

When you import a package, Python looks for a file named `__init__.py` in the package directory. If it finds it, it executes the code in the file. This file is used to initialize the package and to define the package's `__all__` attribute.

### 10.3.2 `__init__.py`

package mypackage  
package mypackage  
package mypackage

```
mypackage/  
  __init__.py  
  A/  
    __init__.py  
    spam.py  
    grok.py  
  B/  
    __init__.py  
    bar.py
```

package mypackage.A.spam  
package grok  
import

```
# mypackage/A/spam.py  
  
from . import  
grok
```

package mypackage.A.spam  
package B.bar  
import

```
# mypackage/A/spam.py
```

```
from ..B import  
bar
```

```
import spam.py  
from spam import *
```

### 10.3.3

```
from mypackage.A import  
grok  
from . import  
grok  
import grok  
# Error (not found)
```



Python 3.6.0 (tags/python-3.6.0:0a70019, Dec 14 2016, 00:00:00) [AMD64]  
Python 3.6.0 (tags/python-3.6.0:0a70019, Dec 14 2016, 00:00:00) [AMD64]

```
% python3 mypackage/A/spam.py      # Relative imports fail
```

Python 3.6.0 (tags/python-3.6.0:0a70019, Dec 14 2016, 00:00:00) [AMD64]  
Python 3.6.0 (tags/python-3.6.0:0a70019, Dec 14 2016, 00:00:00) [AMD64]

```
% python3 -m mypackage.A.spam      # Relative imports work
```

Python 3.6.0 (tags/python-3.6.0:0a70019, Dec 14 2016, 00:00:00) [AMD64]  
[http://www.python.org/dev/peps/ pep-0328](http://www.python.org/dev/peps/ pep-0328)

## 10.4 Python 3.6.0

### 10.4.1 Python 3.6.0

Python 3.6.0 (tags/python-3.6.0:0a70019, Dec 14 2016, 00:00:00) [AMD64]  
Python 3.6.0 (tags/python-3.6.0:0a70019, Dec 14 2016, 00:00:00) [AMD64]  
Python 3.6.0 (tags/python-3.6.0:0a70019, Dec 14 2016, 00:00:00) [AMD64]

### 10.4.2 Python 3.6.0



□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□

```
# mymodule.py

class A
:
    def
spam(self):
    print
('A.spam')

class B
(A):
    def
bar(self):
    print
('B.bar')
```

□□□□ *mymodule.py* □□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□ *mymodule.py* □□□□□  
*mymodule* □□□□□□□□□□□□□□□□□□□□□□□□

```
mymodule/
    __init__.py
    a.py
    b.py
```

□□□ *a.py* □□□□□□□□□□

```
# a.py

class A
:
    def
spam(self):
    print
('A.spam')
```

□□□□ *b.py* □□□□□□□□□□

```
# b.py

from .a import
A

class B
(A):
    def
bar(self):
    print
('B.bar')
```

\_\_init\_\_.py

```
# __init__.py

from .a import
A
from .b import
B
```

mypackage

```
>>> import mymodule

>>> a = mymodule.A()
>>> a.spam()
A.spam
>>> b = mymodule.B()
>>> b.bar()
B.bar
>>>
```

## 10.4.3

from mymodule.a import  
A

```
from mymodule.a import  
A  
from mymodule.b import  
B  
...
```

from mymodule import  
A, B

```
from mymodule import  
A, B
```

from mymodule import  
A, B

```
class B  
class A  
from .a  
import A  
class A
```



class A: class B:

```
>>> import mymodule
```

```
>>> a = mymodule.A()
```

```
>>> a.spam()
```

```
A.spam
```

```
>>>
```

```
if
```

```
isinstance(x, mymodule.A):          # Error
```

```
...
```

```
if
```

```
isinstance(x, mymodule.a.A):        # Ok
```

```
...
```

multiprocessing/\_\_init\_\_.py

## 10.5 包和子包

### 10.5.1 包

包是一个目录，其中包含一个或多个模块。包名通常由点分隔的单词组成，例如 `foo.bar`。包名中的点表示包的层次结构。包名中的点也可以用来分隔包的版本信息，例如 `foo.bar.1.0`。

### 10.5.2 包结构

包的目录结构如下所示。Python 解释器在搜索模块时，会按照这个顺序进行查找。

包的目录结构如下所示。Python 解释器在搜索模块时，会按照这个顺序进行查找。  
`__init__.py` 文件是包的入口点，它定义了包的名称和版本信息。

|                                  |  |
|----------------------------------|--|
| foo-package/<br>spam/<br>blah.py |  |
| bar-package/<br>spam/            |  |

```
grok.py
```

spam.py 文件在 spam 包中，而 \_\_init\_\_.py 文件在包中。

foo-package 和 bar-package 包在 Python 包中。

```
>>> import sys

>>> sys.path.extend(['foo-package', 'bar-package'])
>>> import spam.blah

>>> import spam.grok

>>>
```

spam.blah 和 spam.grok 包在 spam 包中。

### 10.5.3 包

namespace package 是指一个包，其子包不在同一个包中，而是在不同的包中。namespace package 是指一个包，其子包不在同一个包中，而是在不同的包中。



```

__init__.py
__init__.py
__path__

```

```
>>> spam.__path__
_NamespacePath(['foo-package/spam', 'bar-package/spam'])
>>>
```

```
my-package/  
  spam/  
    custom.py
```

sys.path 的搜索顺序  
spam 模块

```
>>> import spam.custom
```

```
>>> import spam.grok
```

```
>>> import spam.blah
```

```
>>>
```

\_\_file\_\_ 属性  
\_\_namespace\_\_ 属性  
“namespace” 属性

```
>>> spam.__file__
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'module' object has no attribute '__file__'
>>> spam
<module 'spam' (namespace)>
>>>
```

PEP 420  
<http://www.python.org/dev/peps/pep-0420>

## 10.6 模块重载

### 10.6.1 简介

模块重载是指在不重启解释器的情况下，重新加载已经加载过的模块。这在开发过程中非常有用，可以避免每次修改代码后都要重启解释器。

### 10.6.2 使用 imp.reload()

使用 `imp.reload()` 函数可以重载模块。该函数接受一个模块对象作为参数，并返回重载后的模块对象。

```
>>> import spam

>>> import imp

>>> imp.reload(spam)
<module 'spam' from './spam.py'>
>>>
```

### 10.6.3 注意事项

使用 `imp.reload()` 重载模块时，需要注意以下几点：

`reload()` 函数返回一个字典 `__dict__`，字典的键是模块的符号名，值是模块的字典。字典的键是模块的符号名，值是模块的字典。  
`id()` 函数返回模块的标识符。

从模块 `module` 中导入名称 `name`。  
`reload()` 函数返回模块的字典。  
字典

```
# spam.py
```

```
def
```

```
bar():  
    print
```

```
('bar')
```

```
def
```

```
grok():  
    print
```

```
('grok')
```

字典

```
>>> import spam
```

```
>>> from spam import
```

```
grok
```

```
>>> spam.bar()
```

```
bar
>>> grok()
grok
>>>
```

□□□□Python□□□□□*spam.py* □□□□□□  
grok()□□□□□□□□□□

```
def

grok():
    print

('New grok')
```

□□□□□□□□□□*reload()*□□□□□□□□□□

```
>>> import imp

>>> imp.reload(spam)
<module 'spam' from './spam.py'>
>>> spam.bar()
bar
>>> grok()                # Notice old output

grok
>>> spam.grok()           # Notice new output

New grok
>>>
```

grok() Python  
Python

Python

## 10.7 Python zip

### 10.7.1

Python

### 10.7.2

Python  
\_\_main\_\_.py Python

```
myapplication/  
  spam.py  
  bar.py  
  grok.py  
  __main__.py
```

\_\_main\_\_.py Python

```
bash % python3 myapplication
```

□□□□□ *\_\_main\_\_.py* □□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□ zip□□□□□□□□□□□□  
□□

```
bash % ls
spam.py bar.py grok.py __main__.py
bash % zip -r myapp.zip *.py
bash % python3 myapp.zip
... output from __main__.py ...
```

## 10.7.3 □□

□□□□□□□ zip□□□□□□□□□□□□ *\_\_main\_\_.py* □  
□□□□□□□□□□□ Python□□□□□□□□□□□□□□□□  
Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□ zip□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□ shell□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
*myapp.zip* □□□□□□□□□□□□□□□□□□□□□□□□□□

```
#!/usr/bin/env python3 /usr/local/bin/myapp.zip
```

## 10.8 包和包管理

### 10.8.1 包

包是一个目录，其中包含一个或多个模块。包可以包含子包，也可以包含数据文件。包的结构如下：

### 10.8.2 包结构

包结构如下：

```
mypackage/  
  __init__.py  
  somedata.dat  
  spam.py
```

在包中，`spam.py` 可以访问 `somedata.dat` 文件。包的结构如下：

```
# spam.py  
  
import pkgutil  
  
data = pkgutil.get_data(__package__, 'somedata.dat')
```



data 文件  
文件

### 10.8.3 文件

文件 I/O  
open() 函数

文件 I/O 模式  
\_\_file\_\_ 变量

.zip 和 .egg 文件  
open() 函数  
archive 函数

pkgutil.get\_data() 函数

get\_data() 函数  
\_\_package\_\_ 变量  
UNIX 系统

## 10.9 sys.path

## 10.9.1 简介

Python 的 `sys.path` 变量包含 Python 解释器搜索模块的目录列表。

## 10.9.2 环境变量

通过设置 `sys.path` 环境变量 `PYTHONPATH` 来指定搜索目录。

```
bash % env PYTHONPATH=/some/dir:/other/dir python3
Python 3.3.0 (default, Oct 4 2012, 10:17:33)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> import sys
>>> sys.path
['', '/some/dir', '/other/dir', ...]
>>>
```

在 shell 中设置环境变量：

创建 `.pth` 文件来指定搜索目录。

```
# myapplication.pth
/some/dir
/other/dir
```



```
import sys
```

```
from os.path import
```

```
abspath, join, dirname
```

```
sys.path.insert(0, abspath(dirname('__file__'), 'src'))
```

src sys.path src  
src sys.path

*site-packages* src  
src  
*.pth* *site-packages* src  
src  
src *.pth*

## 10.10 src

### 10.10.1

src  
src import

### 10.10.2

importlib.import\_module()은 importlib 모듈의  
import\_module() 메서드를 호출하여 모듈을 가져옵니다.

```
>>> import importlib

>>> math = importlib.import_module('math')
>>> math.sin(2)
0.9092974268256817
>>> mod = importlib.import_module('urllib.request')
>>> u = mod.urlopen('http://www.python.org')
>>>
```

import\_module()은 importlib 모듈의  
import\_module() 메서드를 호출하여 모듈을 가져옵니다.

importlib.import\_module()은 importlib 모듈의  
import\_module() 메서드를 호출하여 모듈을 가져옵니다.

```
import importlib

# Same as 'from . import b'

b = importlib.import_module('.b', __package__)
```

## 10.10.3 importlib

```
import_module()은 모듈을 가져오는 데 사용됩니다.
이 함수는 모듈의 이름을 문자열로 제공하고,
해당 모듈을 가져옵니다.
```

```
__import__()은 모듈을 가져오는 데 사용됩니다.
이 함수는 모듈의 이름을 문자열로 제공하고,
importlib.import_module()을 호출하여
모듈을 가져옵니다.
```

10.11 모듈을 가져오는 방법

## 10.11 import 모듈을 가져오는 방법

### 10.11.1 import

Python에서 import를 사용하여 모듈을 가져옵니다.
이 함수는 모듈의 이름을 문자열로 제공하고,
해당 모듈을 가져옵니다.

### 10.11.2 from

Python에서 from를 사용하여 모듈의 특정 부분만 가져옵니다.
이 함수는 모듈의 이름을 문자열로 제공하고,
해당 모듈의 특정 부분을 가져옵니다.
Python에서 import를 사용하여 모듈을 가져옵니다.
이 함수는 모듈의 이름을 문자열로 제공하고,
해당 모듈을 가져옵니다.

import sys  
sys.path.append('testcode/')

import sys  
sys.path.append('testcode/')  
Python

```
testcode/  
  spam.py  
  fib.py  
  grok/  
    __init__.py  
    blah.py
```

```
# spam.py  
  
print  
('I'm spam')  
  
def  
hello(name):  
    print  
('Hello %s' % name)  
  
# fib.py  
  
print  
('I'm fib')
```





Python urllib

```
>>> from urllib.request import
urlopen
>>> u = urlopen('http://localhost:15000/fib.py')
>>> data = u.read().decode('utf-8')
>>> print
(data)
# fib.py
print("I'm fib")
def fib(n):
    if n < 2:
        return 1
    else:
        return fib(n-1) + fib(n-2)
>>>
```

urlopen() import

```
import imp
import urllib.request
import sys
```

**def**

```
load_module(url):
    u = urllib.request.urlopen(url)
    source = u.read().decode('utf-8')
    mod = sys.modules.setdefault(url, imp.new_module(url))
    code = compile(source, url, 'exec')
    mod.__file__ = url
    mod.__package__ = ''
    exec
```

```
(code, mod.__dict__)
    return
```

mod

compile() code

```
>>> fib = load_module('http://localhost:15000/fib.py')
I'm fib
>>> fib.fib(10)
89
>>> spam = load_module('http://localhost:15000/spam.py')
I'm spam
>>> spam.hello('Guido')
Hello Guido
>>> fib
<module 'http://localhost:15000/fib.py' from
'http://localhost:15000/fib.py'>
>>> spam
<module 'http://localhost:15000/spam.py' from
'http://localhost:15000/spam.py'>
>>>
```



```
# Get links from a given URL
```

```
def
```

```
    _get_links(url):
```

```
        class LinkParser
```

```
            (HTMLParser):
```

```
                def
```

```
                    handle_starttag(self, tag, attrs):
```

```
                        if
```

```
                            tag == 'a':
```

```
                                attrs = dict(attrs)
```

```
                                links.add(attrs.get('href').rstrip('/'))
```

```
                            links = set()
```

```
                            try
```

```
                                :
```

```
                                    log.debug('Getting links from %s' % url)
```

```
                                    u = urlopen(url)
```

```
                                    parser = LinkParser()
```

```
                                    parser.feed(u.read().decode('utf-8'))
```

```
                                except Exception as
```

```
                                    e:
```

```
                                        log.debug('Could not get links. %s', e)
```

```
                                        log.debug('links: %r', links)
```

```
                                        return
```

```
links
```

```
class UrlMetaFinder
```

```
    (importlib.abc.MetaPathFinder):
```

```
        def
```

```
            __init__(self, baseurl):
```

```
                self._baseurl = baseurl
```

```
                self._links = { }
```

```
                self._loaders = { baseurl : UrlModuleLoader(baseurl) }
```

```

def
find_module(self, fullname, path=None):
    log.debug('find_module: fullname=%r, path=%r',
fullname, path)
    if

path is
None:
    baseurl = self._baseurl
    else

:
    if not
path[0].startswith(self._baseurl):
    return

None
    baseurl = path[0]

    parts = fullname.split('.')
    basename = parts[-1]
    log.debug('find_module: baseurl=%r, basename=%r',
baseurl, basename)

    # Check link cache

    if
basename not in
self._links:
    self._links[baseurl] = _get_links(baseurl)

    # Check if it's a package

    if
basename in
self._links[baseurl]:

```

```

        log.debug('find_module: trying package %r',
fullname)
        fullurl = self._baseurl + '/' + basename
        # Attempt to load the package (which accesses
__init__.py)

        loader = UrlPackageLoader(fullurl)
        try

:
            loader.load_module(fullname)
            self._links[fullurl] = _get_links(fullurl)
            self._loaders[fullurl] =
UrlModuleLoader(fullurl)
            log.debug('find_module: package %r loaded',
fullname)
        except ImportError as

e:
            log.debug('find_module: package failed. %s',
e)
            loader = None
        return

loader

        # A normal module

        filename = basename + '.py'
        if

filename in
self._links[baseurl]:
            log.debug('find_module: module %r found',
fullname)
            return

self._loaders[baseurl]
        else

:
            log.debug('find_module: module %r not found',

```

```

fullname)
        return

None

    def

invalidate_caches(self):
    log.debug('invalidating link cache')
    self._links.clear()

# Module Loader for a URL

class UrlModuleLoader

(importlib.abc.SourceLoader):
    def

    __init__(self, baseurl):
        self._baseurl = baseurl
        self._source_cache = {}

    def

module_repr(self, module):
    return

'<urlmodule %r from %r>' % (module.__name__, module.__file__)

    # Required method

    def

load_module(self, fullname):
    code = self.get_code(fullname)
    mod = sys.modules.setdefault(fullname,
imp.new_module(fullname))
    mod.__file__ = self.get_filename(fullname)
    mod.__loader__ = self
    mod.__package__ = fullname.rpartition('.')[0]
    exec

(code, mod.__dict__)

```

```

        return

mod

    # Optional extensions

    def

get_code(self, fullname):
    src = self.get_source(fullname)
    return

compile(src, self.get_filename(fullname), 'exec')

    def

get_data(self, path):
    pass

    def

get_filename(self, fullname):
    return

self._baseurl + '/' + fullname.split('.')[ -1] + '.py'

    def

get_source(self, fullname):
    filename = self.get_filename(fullname)
    log.debug('loader: reading %r', filename)
    if

filename in

self._source_cache:
    log.debug('loader: cached %r', filename)
    return

self._source_cache[filename]
    try

```



```

:
    u = urlopen(filename)
    source = u.read().decode('utf-8')
    log.debug('loader: %r loaded', filename)
    self._source_cache[filename] = source
    return

source
    except

(HTTPError, URLError) as

e:
    log.debug('loader: %r failed. %s', filename, e)
    raise ImportError

("Can't load %s" % filename)

    def

is_package(self, fullname):
    return

False

# Package loader for a URL

class UrlPackageLoader

(UrlModuleLoader):
    def

load_module(self, fullname):
    mod = super().load_module(fullname)
    mod.__path__ = [ self._baseurl ]
    mod.__package__ = fullname

    def

get_filename(self, fullname):
    return

self._baseurl + '/' + '__init__.py'

```



```

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named 'fib'

>>> # Load the importer and retry (it works)

>>> import urlimport

>>> urlimport.install_meta('http://localhost:15000')
>>> import fib

I'm fib
>>> import spam

I'm spam
>>> import grok.blah

I'm grok.__init__
I'm grok.blah
>>> grok.blah.__file__
'http://localhost:15000/grok/blah.py'
>>>

```

```

    _____
UrlMetaFinder sys.meta_path
    sys.meta_path
    UrlMetaFinder

```

UrlMetaFinder URL  
URL  
UrlModuleLoader  
HTTP

hook  
sys.path  
*urlimport.py*

```
# urlimport.py

# ... include previous code above ...

# Path finder class for a URL

class UrlPathFinder

(importlib.abc.PathEntryFinder):
    def

    __init__(self, baseurl):
        self._links = None
        self._loader = UrlModuleLoader(baseurl)
        self._baseurl = baseurl

    def

    find_loader(self, fullname):
        log.debug('find_loader: %r', fullname)
```



```

(loader, [fullurl])

    # A normal module

    filename = basename + '.py'
    if
filename in
self._links:
    log.debug('find_loader: module %r found',
fullname)
        return
(self._loader, [])
    else

:
        log.debug('find_loader: module %r not found',
fullname)
        return
(None, [])

    def

invalidate_caches(self):
    log.debug('invalidating link cache')
    self._links = None

# Check path to see if it looks like a URL

_url_path_cache = {}
def
handle_url(path):
    if
path.startswith(('http://', 'https://')):
    log.debug('Handle path? %s. [Yes]', path)
    if
path in

```

```

_url_path_cache:
    finder = _url_path_cache[path]
    else

:
    finder = UrlPathFinder(path)
    _url_path_cache[path] = finder
    return

finder
    else

:
    log.debug('Handle path? %s. [No]', path)

def

install_path_hook():
    sys.path_hooks.append(handle_url)
    sys.path_importer_cache.clear()
    log.debug('Installing handle_url')

def

remove_path_hook():
    sys.path_hooks.remove(handle_url)
    sys.path_importer_cache.clear()
    log.debug('Removing handle_url')

```

1. URL sys.path
 2.

```
>>> # Initial import fails
```

```
>>> import fib
```

```

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>

```

```

ImportError: No module named 'fib'

>>> # Install the path hook

>>> import urlimport

>>> urlimport.install_path_hook()

>>> # Imports still fail (not on path)

>>> import fib

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named 'fib'

>>> # Add an entry to sys.path and watch it work

>>> import sys

>>> sys.path.append('http://localhost:15000')
>>> import fib

I'm fib
>>> import grok.blah

I'm grok.__init__
I'm grok.blah
>>> grok.blah.__file__
'http://localhost:15000/grok/blah.py'
>>>

```

□□□□□□□□□□handle\_url()□□□□□□□□□□  
 sys.path\_hooks□□□□□□□□sys.path□□□□□□□□



sys.path\_hooks[] finder object[]  
sys.path[]

[]  
[]

```
>>> fib
<urlmodule 'fib' from
'http
://localhost
:15000/fib.py
'>
>>> fib.__name__
'fib'
>>> fib.__file__
'http://localhost:15000/fib.py'
>>> import inspect

>>> print
(inspect.getsource(fib))
# fib.py

print
("I'm fib")

def
fib(n):
    if
n < 2:
```

```

        return
1      else
:        return
fib(n-1) + fib(n-2)
>>>

```

### 10.11.3 ☐☐

Python  
Python  
importlib  
<http://docs.python.org/3/library/importlib.html>  
PEP 302  
<http://www.python.org/dev/peps/pep-0302>

```

module object
imp.new module()

```

```
>>> import imp

>>> m = imp.new_module('spam')
>>> m
<module 'spam'>
>>> m.__name__
'spam'
```

```
>>>
```

```
__file__
__package__
```

```
sys.modules
```

```
>>> import sys
```

```
>>> import imp
```

```
>>> m = sys.modules.setdefault('spam', imp.new_module('spam'))
>>> m
<module 'spam'>
>>>
```

```
>>> import math
```

```
>>> m = sys.modules.setdefault('math', imp.new_module('math'))
>>> m
<module 'math' from '/usr/local/lib/python3.3/lib-
dynload/math.so'>
>>> m.sin(2)
0.9092974268256817
>>> m.cos(2)
-0.4161468365471424
```

>>>

```

import sys
sys.path.append('..')
from load_module import load_module
import __init__.py

```

```

import

```

```
from importlib.metadata import version
import sys
sys.meta_path
```

```
>>> from pprint import
pprint
>>> pprint(sys.meta_path)
[<class '_frozen_importlib.BuiltinImporter'>,
 <class '_frozen_importlib.FrozenImporter'>,
 <class '_frozen_importlib.PathFinder'>]
>>>
```

```
import fib
sys.meta_path
find_module()
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□

```
>>> class Finder
:
...
    def
find_module(self, fullname, path):
...
    print
('Looking for', fullname, path)
...
    return
None
...

>>> import sys

>>> sys.meta_path.insert(0, Finder()) # Insert as first entry

>>> import math

Looking for math None
>>> import types

Looking for types None
>>> import threading

Looking for threading None
Looking for time None
```

```
Looking for traceback None
Looking for linecache None
Looking for tokenize None
Looking for token None
>>>
```

```
import sys
sys.path.append(sys._path_)
sys.path.append(sys._path_)
sys.path.append(sys._path_)
xml.etree.ElementTree
```

```
>>> import xml.etree.ElementTree
```

```
Looking for xml None
Looking for xml.etree ['/usr/local/lib/python3.3/xml']
Looking for xml.etree.ElementTree
['/usr/local/lib/python3.3/xml/etree']
Looking for warnings None
Looking for contextlib None
Looking for xml.etree.ElementPath
['/usr/local/lib/python3.3/xml/etree']
Looking for _elementtree None
Looking for copy None
Looking for org None
Looking for pyexpat None
Looking for ElementC14N None
>>>
```

```
sys.meta_path
```

```
>>> del
```

```
>>> import datetime
```

```
>>> import fib
```

```
Looking for xml.superfast ['/usr/local/lib/python3.3/xml']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named 'xml.superfast'
>>>
```

```

    UrlMetaFinder sys.meta_path
    UrlMetaFinder sys.meta_path
    path

```

URL  
URL

UrlPackageLoader  
\_\_init\_\_.py  
\_\_path\_\_  
find\_module()

import  
sys.path  
Python

```
>>> from pprint import
pprint
>>> import sys

>>> pprint(sys.path)
['',
 '/usr/local/lib/python3.3.zip',
 '/usr/local/lib/python3.3',
 '/usr/local/lib/python3.3/plat-darwin',
 '/usr/local/lib/python3.3/lib-dynload',
 '/usr/local/lib/...3.3/site-packages']
>>>
```

sys.path  
sys.path\_importer\_cache

```
>>> pprint(sys.path_importer_cache)
{'.': FileFinder('.')},
```



```

'/usr/local/lib/python3.3':
FileFinder('/usr/local/lib/python3.3'),
'/usr/local/lib/python3.3/':
FileFinder('/usr/local/lib/python3.3/'),
'/usr/local/lib/python3.3/collections':
FileFinder('...python3.3/collections'),
'/usr/local/lib/python3.3/encodings':
FileFinder('...python3.3/encodings'),
'/usr/local/lib/python3.3/lib-dynload':
FileFinder('...python3.3/lib-dynload'),
'/usr/local/lib/python3.3/plat-darwin':
FileFinder('...python3.3/plat-darwin'),
'/usr/local/lib/python3.3/site-packages':
FileFinder('...python3.3/site-packages'),
'/usr/local/lib/python3.3.zip': None}
>>>

```

```

sys.path_importer_cache[sys.path[0]]
[...]
```

```

import fib
sys.path[0] = fib
sys.path_importer_cache[sys.path[0]]
[...]
```

```

>>> class Finder
:
...
    def
find_loader(self, name):
...

```

```

        print
('Looking for', name)
...

        return

(None, [])
...

>>> import sys

>>> # Add a "debug" entry to the importer cache

>>> sys.path_importer_cache['debug'] = Finder()
>>> # Add a "debug" directory to sys.path

>>> sys.path.insert(0, 'debug')
>>> import threading

Looking for threading
Looking for time
Looking for traceback
Looking for linecache
Looking for tokenize
Looking for token
>>>

```

```

debugsys.pathdebug(None,
[])

```

sys.path\_importer\_cache  
sys.path\_hooks  
sys.path\_hooks

```
>>> sys.path_importer_cache.clear()
>>>

def
check_path(path):
...

    print
('Checking', path)
...

    raise ImportError

()
...

>>> sys.path_hooks.insert(0, check_path)
>>> import fib

Checked debug
Checking .
Checking /usr/local/lib/python3.3.zip
Checking /usr/local/lib/python3.3
Checking /usr/local/lib/python3.3/plat-darwin
Checking /usr/local/lib/python3.3/lib-dynload
Checking /Users/beazley/.local/lib/python3.3/site-packages
Checking /usr/local/lib/python3.3/site-packages
Looking for fib
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named 'fib'
>>>
```

```
sys.path
check_path()
ImportError
sys.path_hooks
```

```
sys.path
URL
```

```
>>> def
check_url(path):
...
    if
path.startswith('http://'):
...
        return
Finder()
...
    else
:
...
        raise ImportError
()
...

>>> sys.path.append('http://localhost:15000')
>>> sys.path_hooks[0] = check_url
>>> import fib
```

```

Looking for fib                                # Finder output!
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named 'fib'

>>> # Notice installation of Finder in sys.path_importer_cache

>>> sys.path_importer_cache['http://localhost:15000']
<__main__.Finder object at 0x10064c850>
>>>

```

```

sys.path_hooks
sys.path_importer_cache
sys.path

```

```

find_loader()
find_loader(loader, None)
loader

```

```

find_loader()
(loader, path)
loader
__init__.py
path
URL
http://localhost:15000
import
grok
find_loader()

```

```
['[http://localhost: 15000/grok]
(http://localhost: 15000/grok)']
```

```
find_loader()10.5
__init__.py find_loader()
(None, path)path
__path__
__init__.py import
sys.path
10.5
```

```
__path__
```

```
>>> import xml.etree.ElementTree

>>> xml.__path__
['/usr/local/lib/python3.3/xml']
>>> xml.etree.__path__
['/usr/local/lib/python3.3/xml/etree']
>>>
```

```
__path__find_loader()
__path__
sys.path_hooks
```

```
__path__ = None
def handle_url():
    """
    Returns the URL path finder for the current URL.
    """
    sys.path_importer_cache = {}
```

```
def handle_url():
    """
    Returns the URL path finder for the current URL.
    """
    # Check link cache
    if self._links is None:
        self._links = [] # See discussion
    self._links = _get_links(self._baseurl)
    # URL
    return self._links
```

```
# Check link cache

if
self._links is
None:
    self._links = [] # See discussion

    self._links = _get_links(self._baseurl)
```

```
def handle_url():
    """
    Returns the URL path finder for the current URL.
    """
```

```
importlib.invalidate_caches()
importlib.invalidate_caches()
URL
```

```
sys.meta_path
sys.meta_path
bookkeeping
UrlMetaFinder
sys.path
sys.path
```

```
>>> import logging

>>> logging.basicConfig(level=logging.DEBUG)
>>> import urlimport

>>> urlimport.install_path_hook()
DEBUG:urlimport:Installing handle_url
>>> import fib
```



```

DEBUG:urlimport:Handle path? /usr/local/lib/python33.zip. [No]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named 'fib'
>>> import sys

>>> sys.path.append('http://localhost:15000')
>>> import fib

DEBUG:urlimport:Handle path? http://localhost:15000. [Yes]
DEBUG:urlimport:Getting links from http://localhost:15000
DEBUG:urlimport:links: {'spam.py', 'fib.py', 'grok'}
DEBUG:urlimport:find_loader: 'fib'
DEBUG:urlimport:find_loader: module 'fib' found
DEBUG:urlimport:loader: reading
'http://localhost:15000/fib.py'
DEBUG:urlimport:loader: 'http://localhost:15000/fib.py' loaded
I'm fib
>>>

```

PEP 302  
<http://www.python.org/dev/peps/pep-0302>

## 10.12

### 10.12.1

importlib



```

self._skip:
    return

None
    self._skip.add(fullname)
    return

PostImportLoader(self)

class PostImportLoader
:
    def

__init__(self, finder):
    self._finder = finder

    def

load_module(self, fullname):
    importlib.import_module(fullname)
    module = sys.modules[fullname]
    for

func in

_post_import_hooks[fullname]:
    func(module)
    self._finder._skip.remove(fullname)
    return

module

def

when_imported(fullname):
    def

decorate(func):
    if

fullname in

sys.modules:

```

```

        func(sys.modules[fullname])
    else
:
        _post_import_hooks[fullname].append(func)
    return
func
    return
decorate
sys.meta_path.insert(0, PostImportFinder())

```

when\_imported()

```

>>> from postimport import
when_imported
>>> @when_imported('threading')
... def
warn_threads(mod):
...     print
('Threads? Are you crazy?')
...
>>>
>>> import threading

Threads? Are you crazy?
>>>

```

when\_imported()

```

from functools import
wraps
from postimport import
when_imported
def
logged(func):
    @wraps(func)
    def
wrapper(*args, **kwargs):
    print
('Calling', func.__name__, args, kwargs)
    return
func(*args, **kwargs)
    return
wrapper
# Example
@when_imported('math')
def
add_logging(mod):
    mod.cos = logged(mod.cos)
    mod.sin = logged(mod.sin)

```

## 10.12.3 □□

□□□□□10.11□□□□□□import□□□□□□□□□□  
□□

```
    @when_imported
    sys.modules
    _post_import_hooks
    _post_import_hooks
    
```

```
    _post_import_hooks
    PostImportFinder
    sys.meta_path 10.11
    sys.meta_path
    PostImportFinder
    
```

```
    PostImportFinder
    sys.meta_path
    PostImportLoader
    imp.import_module()
    PostImportFinder
    PostImportFinder
    import
    sys.meta_path
    
```

imp.import\_module() 函数在导入模块时，会先调用该模块的 `__post_import_hooks__` 属性中的函数，然后再执行模块的 `__init__` 方法。

在 `__post_import_hooks__` 属性中，可以定义一个或多个函数，这些函数会在模块导入完成后被调用。例如，可以在 `@when_imported()` 装饰器中定义一个函数，该函数会在模块被导入时自动执行。

`imp.reload()` 函数用于重新加载模块。它接受一个模块对象作为参数，并返回该模块的重新加载后的对象。在 `sys.modules` 字典中，模块的键值对会被更新，以反映重新加载后的模块。

在 `post-import` 部分，PEP 369 提出了一种新的 `importlib` 模块，用于更灵活地管理模块的导入。该模块提供了一种新的 `importlib` 模块，用于更灵活地管理模块的导入。

## 10.13 模块的导入

### 10.13.1 模块

Python

## 10.13.2 ☐☐☐☐

```
Python
~./local/lib/python3.3/site-packages
--user

```

```
python3 setup.py install --user
```

11

```
pip install --user packagename
```

```

    site-package sys.path
    site-package
    distribute pip

```

### 10.13.3 □□



□□□□□□□□□□□□□□ *site-package* □□□□□  
□□□□/usr/local/lib/ *python3.3/site-packages*  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□sudo□□□□□□□□□□□□□□□□□□□□□□□□sudo□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□

## 10.14 □□□□Python□□

### 10.14.1 □□

□□□□□□□□□Python□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□Python□□□□□□□□□□□  
□□□□□□□Python□□□□□□□

### 10.14.2 □□□□

□□□□pyenv□□□□□□□□□“□□”□□□□□□□□□□□□  
□□Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
*Scripts* □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
bash % pyvenv Spam
bash %
```

pyvenv 在指定的目录中创建 Python 虚拟环境。在指定的目录中创建 Python 虚拟环境。在指定的目录中创建 Python 虚拟环境。

*Spam* 在指定的目录中创建 Python 虚拟环境。

```
bash % cd Spam
bash % ls
bin          include          lib          pyvenv.cfg
bash %
```

bin 目录包含 Python 解释器和标准库。bin 目录包含 Python 解释器和标准库。bin 目录包含 Python 解释器和标准库。

```
bash % Spam/bin/python3
Python 3.3.0 (default, Oct 6 2012, 15:45:22)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> from pprint import pprint
>>> import sys
>>> pprint(sys.path)
['',
 '/usr/local/lib/python33.zip',
 '/usr/local/lib/python3.3',
 '/usr/local/lib/python3.3/plat-darwin',
 '/usr/local/lib/python3.3/lib-dynload',
 '/Users/beazley/Spam/lib/python3.3/site-packages']
>>>
```

在指定的目录中创建 Python 虚拟环境。在指定的目录中创建 Python 虚拟环境。在指定的目录中创建 Python 虚拟环境。

在指定的目录中创建 Python 虚拟环境。在指定的目录中创建 Python 虚拟环境。在指定的目录中创建 Python 虚拟环境。

在指定的目录中创建 Python 虚拟环境。在指定的目录中创建 Python 虚拟环境。在指定的目录中创建 Python 虚拟环境。

## 10.14.3 虚拟环境

虚拟环境（virtual environment）是一个可以安装和管理 Python 包的环境。它使用 `sys.path` 来管理 Python 包的路径，并允许在 `site-package` 目录下安装和管理包。

虚拟环境通常使用 `distribute` 或 `pip` 来创建和管理。在创建虚拟环境时，可以在 `site-packages` 目录下安装和管理包。

虚拟环境通常使用 Python 来创建和管理。在创建虚拟环境时，可以在 `Python` 目录下安装和管理包。在创建虚拟环境时，可以在 `Python` 目录下安装和管理包。

虚拟环境通常使用 `--system-site-packages` 来创建和管理。在创建虚拟环境时，可以在 `system-site-packages` 目录下安装和管理包。

```
bash % pyvenv --system-site-packages Spam
bash %
```

虚拟环境通常使用 `pyvenv` 来创建和管理。在创建虚拟环境时，可以在 `PEP 405` 网站上找到更多信息：  
[http://www.python.org/dev/peps/ pep-0405](http://www.python.org/dev/peps/pep-0405)

**10.15** □□□□□□□

## 10.15.1 ☐ ☐

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

## 10.15.2 ☐☐☐☐

```
projectname/
  README.txt
  Doc/
    documentation.txt
projectname/
  __init__.py
  foo.py
  bar.py
  utils/
    __init__.py
    spam.py
    grok.py
examples/
  helloworld.py
  ...
```

□□□□□□□□□□□□□□□□ *setup.py* □□□□□□□□  
□□□□

```
# setup.py
from distutils.core import setup
```

```
setup(name='projectname',
      version='1.0',
      author='Your Name',
      author_email='you@youraddress.com',
      url='http://www.you.com/projectname',
      packages=['projectname', 'projectname.utils'],
)
```

MANIFEST.in

```
# MANIFEST.in
include *.txt
recursive-include examples *
recursive-include Doc *
```

□□*setup.py* □*MANIFEST.in* □□□□□□□□□□

```
% bash python3 setup.py sdist
```

`projectname-1.0.zip` `projectname-1.0.tar.gz`

Python Package Index <http://pypi.python.org>

### 10.15.3 □□

```

Python
setup.py
packages=
['projectname', 'projectname.utils']

```

```

Python
distutils
Python 3

```

□□□□□□C□□□□□□□□□□□□□□□□15□□□C  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□15.2□□

# 11 Web

A 4x20 grid of squares. The word 'Python' is displayed in various styles and positions across the grid:

- Row 1: 'Python' in a bold, black, sans-serif font, centered horizontally.
- Row 2: 'Python' in a bold, black, sans-serif font, left-aligned.
- Row 3: 'Python' in a bold, black, sans-serif font, left-aligned.
- Row 4: 'Python' in a bold, black, sans-serif font, left-aligned.

## 11.1 HTTP

### 11.1.1 ☐☐

□□□□□□□□□□□□ HTTP □□□□□□□□□□□□□□□□  
□□□□□□□□ REST API □□□□□

## 11.1.2 ☐ ☐ ☐ ☐

```

    urllib.request.urlopen(
        HTTP GET
    )

```

```
from urllib import
request, parse
# Base URL being accessed
```

```

url = 'http://httpbin.org/get'

# Dictionary of query parameters (if any)

parms = {
    'name1' : 'value1',
    'name2' : 'value2'
}

# Encode the query string

querystring = parse.urlencode(parms)
# Make a GET request and read the response

u = request.urlopen(url+'?' + querystring)
resp = u.read()

```

POST request body  
 urlopen()

```

from urllib import
request, parse

# Base URL being accessed

url = 'http://httpbin.org/post'

# Dictionary of query parameters (if any)

parms = {

```



```

    'name1' : 'value1',
    'name2' : 'value2'
}

# Encode the query string

querystring = parse.urlencode(parms)

# Make a POST request and read the response

u = request.urlopen(url, querystring.encode('ascii'))
resp = u.read()

```

urllib2.urlopen() HTTP request  
 user-agent header  
 Request.urlopen()

```

from urllib import
request, parse
...

# Extra headers

headers = {
    'User-agent' : 'none/ofyourbusiness',
    'Spam' : 'Eggs'
}

req = request.Request(url, querystring.encode('ascii'),
headers=headers)

# Make a request and read the response

```

```
u = request.urlopen(req)
resp = u.read()
```

requests[http://pypi.python.org/pypi/requests](http://pypi.python.org/pypi/requests) [1]

```
import requests

# Base URL being accessed

url = 'http://httpbin.org/post'
# Dictionary of query parameters (if any)

parms = {
    'name1' : 'value1',
    'name2' : 'value2'
}

# Extra headers

headers = {
    'User-agent' : 'none/ofyourbusiness',
    'Spam' : 'Eggs'
}

resp = requests.post(url, data=parms, headers=headers)

# Decoded text returned by the request
```

```
text = resp.text
```

requests 返回的响应对象 `resp` 包含以下属性：

- `resp.text`：响应的文本内容，编码为 `Unicode`。
- `resp.content`：响应的二进制内容。
- `resp.json`：如果响应是 JSON 格式，则返回解析后的 Python 字典。

使用 `requests.head()` 方法可以获取响应的头部信息，返回一个 `HTTPResponse` 对象。

```
import requests
```

```
resp = requests.head('http://www.python.org/index.html')

status = resp.status_code
last_modified = resp.headers['last-modified']
content_type = resp.headers['content-type']
content_length = resp.headers['content-length']
```

requests 是一个 Python Package Index (PyPI) 上的开源库。

```
import requests
```

```
resp =
requests.get('http://pypi.python.org/pypi?action=login',
              auth=('user', 'password'))
```

requests 라이브러리 HTTP  
cookies 사용하기

```
import requests

# First request

resp1 = requests.get(url)
...
# Second requests with cookies received on first requests

resp2 = requests.get(url, cookies=resp1.cookies)
```

requests 라이브러리  
파일 업로드

```
import requests

url = 'http://httpbin.org/post'
files = { 'file': ('data.csv', open('data.csv', 'rb')) }

r = requests.post(url, files=files)
```

## 11.1.3 网络

Python 提供了两个 HTTP 客户端库，分别是 `urllib` 和 `requests`。前者是 Python 标准库的一部分，后者是一个第三方库。前者支持 GET 和 POST 两种方法，后者支持更多的方法。

下面是一个使用 `requests` 库发送 HEAD 请求的例子：

```
from http.client import
HTTPConnection
from urllib import
parse

c = HTTPConnection('www.python.org', 80)
c.request('HEAD', '/index.html')
resp = c.getresponse()

print
('Status', resp.status)
for
name, value in
resp.getheaders():
```

```
print
(name, value)
```

cookies  
urllib  
Python package index

```
import urllib.request

auth = urllib.request.HTTPBasicAuthHandler()
auth.add_password('pypi', 'http://pypi.python.org', 'username', 'password')
opener = urllib.request.build_opener(auth)

r =
urllib.request.Request('http://pypi.python.org/pypi?action=login')
u = opener.open(r)
resp = u.read()

# From here. You can access more pages using opener

...
```

requests

HTTP  
cookies HTTP

python 的 requests 库使用 httpbin 的  
<http://httpbin.org> 提供的接口来测试  
JSON 数据。

```
>>> import requests

>>> r = requests.get('http://httpbin.org/get?name=Dave&n=37',
...                  headers = { 'User-agent': 'goaway/1.0' })
>>> resp = r.json
>>> resp['headers']
{'User-Agent': 'goaway/1.0', 'Content-Length': '', 'Content-
Type': '',
 'Accept-Encoding': 'gzip, deflate, compress', 'Connection':
 'keep-alive', 'Host': 'httpbin.org', 'Accept': '*//*'}
>>> resp['args']
{'name': 'Dave', 'n': '37'}
>>>
```

python 的 requests 库使用 httpbin.org 提供的  
接口来测试 303 重定向。  
HTTP 重定向。

python 的 requests 库使用 HTTP 重定向  
OAuth 的 requests 库。  
<http://docs.python-requests.org> 提供了  
详细的文档。

## 11.2 `socketserver` TCP 서버

### 11.2.1 개요

`socketserver` TCP 서버를 간단하게 작성할 수 있도록 도와주는 모듈

### 11.2.2 예제

`socketserver` TCP 서버를 간단하게 작성할 수 있도록 도와주는 모듈  
예제: `echo` 서버

```
from socketserver import
BaseRequestHandler, TCPServer

class EchoHandler
(BaseRequestHandler):
    def
handle(self):
        print
('Got connection from', self.client_address)
        while
True:
            msg = self.request.recv(8192)
            if not
msg:
                break

            self.request.send(msg)
```



```
if
```

```
__name__ == '__main__':  
    serv = TCPServer('', 20000), EchoHandler)  
    serv.serve_forever()
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
handle()□□□□□□□□□□□□□□□□□□□□request□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
Python□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
>>> from socket import  
  
socket, AF_INET, SOCK_STREAM  
>>> s = socket(AF_INET, SOCK_STREAM)  
>>> s.connect(('localhost', 20000))  
>>> s.send(b'Hello')  
5  
>>> s.recv(8192)  
b'Hello'  
>>>
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□StreamRequestHandler□□□□□□□□□□□□□□□□  
socket□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□



# ForkingTCPServer vs ThreadingTCPServer

```
from socketserver import
ThreadingTCPServer
...
if
__name__ == '__main__':
    serv = ThreadingTCPServer('', 20000), EchoHandler)
    serv.serve_forever()
```

ThreadingTCPServer is a subclass of TCPServer. It uses threading to handle multiple connections simultaneously. This means that the server can handle multiple requests at the same time without blocking the main thread.

ThreadingTCPServer is a subclass of TCPServer. It uses threading to handle multiple connections simultaneously. This means that the server can handle multiple requests at the same time without blocking the main thread.

```
...
if
__name__ == '__main__':
    from threading import
    Thread
    NWORKERS = 16
    serv = TCPServer('', 20000), EchoHandler)
    for
```

```

n in
range(NWORKERS):
    t = Thread(target=serv.serve_forever)
    t.daemon = True
    t.start()
serv.serve_forever()

```

TCP Server  
 socket  
 socket  
 bind\_and\_activate=False

```

if
__name__ == '__main__':
    serv = TCPServer('', 20000), EchoHandler,
bind_and_activate=False)
    # Set up various socket options

    serv.socket.setsockopt(socket.SOL_SOCKET,
socket.SO_REUSEADDR, True)
    # Bind and activate

    serv.server_bind()
    serv.server_activate()
    serv.serve_forever()

```

socket  
 socket

# TCPServer

```
...
if
__name__ == '__main__':
    TCPServer.allow_reuse_address = True
    serv = TCPServer('', 20000), EchoHandler)
    serv.serve_forever()
```

BaseRequestHandler  
StreamRequestHandler  
StreamRequestHandler

```
import socket

class EchoHandler
(StreamRequestHandler):
    # Optional settings (defaults shown)

    timeout = 5 # Timeout on all
socket operations

    rbufsize = -1 # Read buffer size
```

```

wbufsize = 0                                # Write buffer size

    disable_nagle_algorithm = False           # Sets TCP_NODELAY
    socket option

    def
handle(self):
    print
('Got connection from', self.client_address)
    try
:
        for
line in
self.rfile:
        # self.wfile is a file-like object for writing

        self.wfile.write(line)
    except
socket.timeout:
    print
('Timed out!')

```

Python
 HTTP
 XML-RPC
 socketserver
 socket
 socket

```
from socket import
socket, AF_INET, SOCK_STREAM

def
echo_handler(address, client_socket):
    print
    ('Got connection from {}'.format(address))
    while
True:
        msg = client_socket.recv(8192)
        if not
msg:
            break

        client_socket.sendall(msg)
        client_socket.close()

def
echo_server(address, backlog=5):
    sock = socket(AF_INET, SOCK_STREAM)
    sock.bind(address)
    sock.listen(backlog)
    while
True:
        client_socket, client_addr = sock.accept()
        echo_handler(client_addr, client_socket)

if
__name__ == '__main__':
    echo_server('', 20000)
```

## 11.3 `socketserver`UDP

### 11.3.1 `UDP`

`socketserver`UDP

### 11.3.2 `socketserver`

`socketserver`UDP

```
from socketserver import
BaseRequestHandler, UDPServer
import time

class TimeHandler
(BaseRequestHandler):
    def
handle(self):
    print
('Got connection from', self.client_address)
    # Get message and client socket

    msg, sock = self.request
    resp = time.ctime()
    sock.sendto(resp.encode('ascii'), self.client_address)

if
```



```
__name__ == '__main__':
    serv = UDPServer('', 20000), TimeHandler)
    serv.serve_forever()
```

1. 接收客户端发来的数据  
 handle() 接收客户端发来的 request 数据  
 2. 通过 socket 获取 client\_address

3. 通过 Python 的 socket 模块

```
>>> from socket import
socket, AF_INET, SOCK_DGRAM
>>> s = socket(AF_INET, SOCK_DGRAM)
>>> s.sendto(b'', ('localhost', 20000))
0
>>> s.recvfrom(8192)
(b'Wed Aug 15 20:35:08 2012', ('127.0.0.1', 20000))
>>>
```

### 11.3.3 发送

1. 通过 UDP 发送数据  
 2. 通过 socket 的 sendto()

recvfrom() 接收数据 send() 发送数据  
recv() 接收数据  
UDP 协议

UDP 协议与 TCP 协议的区别  
UDP 是无连接的，而 TCP 是面向连接的。  
UDP 的传输效率比 TCP 高，因为 TCP 需要进行三次握手。  
UDP 的数据包大小有限制，而 TCP 的数据包大小没有限制。  
UDP 不支持流量控制，而 TCP 支持流量控制。  
UDP 不支持重传，而 TCP 支持重传。  
UDP 的可靠性比 TCP 低，因为 UDP 的数据包可能会丢失。  
UDP 适用于对实时性要求高的应用，如视频会议、在线游戏等。  
UDP 适用于对可靠性要求高的应用，如文件传输、数据库连接等。

UDPServer 类  
TCP 类  
ForkingUDPServer 类  
ThreadingUDPServer 类

```
from socketserver import
ThreadingUDPServer
...
if
__name__ == '__main__':
    serv = ThreadingUDPServer(('',20000), TimeHandler)
    serv.serve_forever()
```

socket 模块的 UDP 类

```

from socket import
socket, AF_INET, SOCK_DGRAM
import time

def
time_server(address):
    sock = socket(AF_INET, SOCK_DGRAM)
    sock.bind(address)
    while
True:
        msg, addr = sock.recvfrom(8192)
        print
('Got message from', addr)
        resp = time.ctime()
        sock.sendto(resp.encode('ascii'), addr)

if
__name__ == '__main__':
    time_server('', 20000)

```

## 11.4 CIDR IP

### 11.4.1

“123.45.67.89/27” CIDR  
 Classless InterDomain Routing  
 IP

["123.45.67.64", "123.45.67.65" ... "123.45.67.95"]

## 11.4.2 itertools

itertools.ipaddress

```
>>> import itertools

>>> net = itertools.ip_network('123.45.67.64/27')
>>> net
IPv4Network('123.45.67.64/27')
>>> for
a in
net:
...
    print
(a)
...

123.45.67.64
123.45.67.65
123.45.67.66
123.45.67.67
123.45.67.68
...

123.45.67.95
>>>

>>> net6 =
itertools.ip_network('12:3456:78:90ab:cd:ef01:23:30/125')
>>> net6
```

```
IPv6Network('12:3456:78:90ab:cd:ef01:23:30/125')
>>> for
a in
net6:
...
    print
(a)
...
12:3456:78:90ab:cd:ef01:23:30
12:3456:78:90ab:cd:ef01:23:31
12:3456:78:90ab:cd:ef01:23:32
12:3456:78:90ab:cd:ef01:23:33
12:3456:78:90ab:cd:ef01:23:34
12:3456:78:90ab:cd:ef01:23:35
12:3456:78:90ab:cd:ef01:23:36
12:3456:78:90ab:cd:ef01:23:37
>>>
```

network[ ]

```
>>> net.num_addresses
32
>>> net[0]
IPv4Address('123.45.67.64')
>>> net[1]
IPv4Address('123.45.67.65')
>>> net[-1]
IPv4Address('123.45.67.95')
>>> net[-2]
IPv4Address('123.45.67.94')
>>>
```

□□□□□□□□□□□□□□□□

```
>>> a = ipaddress.ip_address('123.45.67.69')
>>> a in
net
True
>>> b = ipaddress.ip_address('123.45.67.123')
>>> b in
net
False
>>>
```

IP□□□□□□□□□□□□□□□□IP□□□interface□□□  
□□

```
>>> inet = ipaddress.ip_interface('123.45.67.73/27')
>>> inet.network
IPv4Network('123.45.67.64/27')
>>> inet.ip
IPv4Address('123.45.67.73')
>>>
```

## 11.4.3 □□

ipaddress□□□□□□□□□□□□IP□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□

ipaddress  
socket  
IPv4Address  
str()

```
>>> a = ipaddress.ip_address('127.0.0.1')
>>> from socket import

socket, AF_INET, SOCK_STREAM
>>> s = socket(AF_INET, SOCK_STREAM)
>>> s.connect((a, 8080))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'IPv4Address' object to str
implicitly
>>> s.connect((str(a), 8080))
>>>
```

“ipaddress”  
”<http://docs.python.org/3/howto/ipaddress.html>

## 11.5 REST

### 11.5.1

REST  
Web

## 11.5.2 小例子

下面是一个REST风格的WSGI应用，  
它实现了PEP 3333中定义的  
<http://www.python.org/dev/peps/pep-3333>  
中定义的接口。

```
# resty.py

import cgi

def
notfound_404(envIRON, start_response):
    start_response('404 Not Found', [ ('Content-type',
    'text/plain') ])
    return
[b'Not Found']

class PathDispatcher

:
    def

__init__(self):
    self.pathmap = { }

    def

__call__(self, environ, start_response):
    path = environ['PATH_INFO']
    params = cgi.FieldStorage(environ['wsgi.input'],
                             environ=environ)
    method = environ['REQUEST_METHOD'].lower()
```





```

        start_response('200 OK', [ ('Content-type', 'text/html')])
        params = environ['params']
        resp = _hello_resp.format(name=params.get('name'))
        yield

resp.encode('utf-8')

_localtime_resp = '''\

<?xml version="1.0"?>
<time>
    <year>{t.tm_year}</year>
    <month>{t.tm_mon}</month>
    <day>{t.tm_mday}</day>
    <hour>{t.tm_hour}</hour>
    <minute>{t.tm_min}</minute>
    <second>{t.tm_sec}</second>
</time>'''

def
localtime(envIRON, start_response):
    start_response('200 OK', [ ('Content-type',
'application/xml') ])
    resp = _localtime_resp.format(t=time.localtime())
    yield

resp.encode('utf-8')

if
__name__ == '__main__':
    from resty import
PathDispatcher
    from wsgiref.simple_server import
make_server

    # Create the dispatcher and register functions

    dispatcher = PathDispatcher()
    dispatcher.register('GET', '/hello', hello_world)

```

```
dispatcher.register('GET', '/localtime', localtime)

# Launch a basic server

httpd = make_server('', 8080, dispatcher)
print
('Serving on port 8080...')
httpd.serve_forever()
```

urllib

```
>>> u = urlopen('http://localhost:8080/hello?name=Guido')
>>> print

(u.read().decode('utf-8'))
<html>
  <head>
    <title>Hello Guido</title>
  </head>
  <body>
    <h1>Hello Guido!</h1>
  </body>
</html>
>>> u = urlopen('http://localhost:8080/localtime')
>>> print

(u.read().decode('utf-8'))
<?xml version="1.0"?>
<time>
  <year>2012</year>
  <month>11</month>
  <day>24</day>
  <hour>14</hour>
  <minute>49</minute>
  <second>17</second>
</time>
>>>
```

## 11.5.3 简介

REST 是建立在 HTTP 协议基础上的，它使用 HTTP 协议中的 GET、POST、PUT、DELETE 等方法来操作数据。REST 使用 XML、JSON、CSV 等格式来传输数据。API 是应用程序编程接口，它是应用程序与外部系统交互的接口。

REST API 是建立在 REST 协议基础上的，它使用 REST 协议中的 GET、POST、PUT、DELETE 等方法来操作数据。REST API 是建立在 REST 协议基础上的，它使用 REST 协议中的 GET、POST、PUT、DELETE 等方法来操作数据。JavaScript、Android、iOS 等应用程序可以通过 REST API 与外部系统交互。

REST 是建立在 Python 的 WSGI 协议基础上的，它使用 WSGI 协议中的 Web 接口来操作数据。WSGI 是 Web 服务网关接口，它是 Web 应用程序与外部系统交互的接口。

WSGI 是 Web 服务网关接口，它是 Web 应用程序与外部系统交互的接口。

```
import cgi
```

```
def
wsgi_app(environ, start_response):
    ...
```

environ 字典包含 WSGI 规范定义的字典，包含 Apache CGI 规范定义的字典

```
def
wsgi_app(environ, start_response):
    method = environ['REQUEST_METHOD']
    path = environ['PATH_INFO']
    # Parse the query parameters

    params = cgi.FieldStorage(environ['wsgi.input'],
environ=environ)
    ...
```

environ 字典包含 WSGI 规范定义的字典，包含 Apache CGI 规范定义的字典

environ['REQUEST\_METHOD'] 包含请求方法，GET、POST、HEAD 等

environ['PATH\_INFO'] 包含请求路径，cgi.FieldStorage() 包含请求参数

```
def start_response(status, headers):
    status, response = HTTPStatus(status).description, status
    headers.append((name, value) for name, value in HTTP_HEADERS.items())
```

```
def
wsgi_app(environ, start_response):
    ...
    start_response('200 OK', [('Content-type', 'text/plain')])
```

WSGI

```
def
wsgi_app(environ, start_response):
    ...
    start_response('200 OK', [('Content-type', 'text/plain')])
    resp = []
    resp.append(b'Hello World\n
')
    resp.append(b'Goodbye!\n
')
    return
resp
```

yield

```

def
wsgi_app(environ, start_response):
    ...
    start_response('200 OK', [('Content-type', 'text/plain')])
    yield
b'Hello World\n
,
    yield
b'Goodbye!\n
,

```

WSGI is a simple interface between web servers and web applications. It defines a standard way for a web server to call a web application, and a standard way for a web application to return a response to a web server.

WSGI is a simple interface between web servers and web applications. It defines a standard way for a web server to call a web application, and a standard way for a web application to return a response to a web server.

```

class WSGIApplication
:
    def
__init__(self):
    ...
    def
__call__(self, environ, start_response)

```

...

```
        PathDispatcher
        environ['params']

```

```
        WSGI
        start_response()

```

```
        print()XML

```

```
        WSGI
        Web WSGI

```



```

if
__name__ == '__main__':
    from wsgiref.simple_server import
make_server

    # Create the dispatcher and register functions

    dispatcher = PathDispatcher()
    ...

    # Launch a basic server

    httpd = make_server('', 8080, dispatcher)
    print
('Serving on port 8080...')
    httpd.serve_forever()

```

WSGI

WSGI

cookies

WebOb <http://webob.org> Paste

<http://pythonpaste.org>

## 11.6 XML-RPC

### 11.6.1

Python

### 11.6.2

XML-RPC

```
from xmlrpc.server import
SimpleXMLRPCServer

class KeyValueServer
:
    _rpc_methods_ = ['get', 'set', 'delete', 'exists', 'keys']
    def

__init__(self, address):
    self._data = {}
    self._serv = SimpleXMLRPCServer(address,
allow_none=True)
    for

name in

self._rpc_methods_:
    self._serv.register_function(getattr(self, name))
```

```

    def
get(self, name):
    return
self._data[name]

    def
set(self, name, value):
    self._data[name] = value

    def
delete(self, name):
    del
self._data[name]

    def
exists(self, name):
    return
name in
self._data

    def
keys(self):
    return
list(self._data)

    def
serve_forever(self):
    self._serv.serve_forever()

# Example

if

```

```
__name__ == '__main__':  
    kvserv = KeyValueServer('', 15000)  
    kvserv.serve_forever()
```

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

```
>>> from xmlrpc.client import
ServerProxy
>>> s = ServerProxy('http://localhost:15000', allow_none=True)
>>> s.set('foo', 'bar')
>>> s.set('spam', [1, 2, 3])
>>> s.keys()
['spam', 'foo']
>>> s.get('foo')
'bar'
>>> s.get('spam')
[1, 2, 3]
>>> s.delete('spam')
>>> s.exists('spam')
False
>>>
```

### 11.6.3 ☐ ☐

```

    """XML-RPC"""
    register_function(serve_forever)
    """
    """

```

```

from xmlrpc.server import
SimpleXMLRPCServer
def
add(x,y):
    return
x+y

serv = SimpleXMLRPCServer('', 15000)
serv.register_function(add)
serv.serve_forever()

```

XML-RPC

```

>>> class Point
:
...
    def
__init__(self, x, y):
...
        self.x = x
...
        self.y = y
...

>>> p = Point(2, 3)
>>> s.set('foo', p)
>>> s.get('foo')

```

```
{'x': 2, 'y': 3}
>>>
```

[illegible]

```
>>> s.set('foo', b'Hello World')
>>> s.get('foo')
<xmlrpc.client.Binary object at 0x10131d410>
>>> _data
b'Hello World'
>>>
```

[illegible]

# XML-RPC

## SimpleXMLRPCServer

### 11.2

#### XML-RPC XML

#### XML-RPC Python

```

    XML-RPC
quick and dirty
XML-RPC

```

## 11.7 子进程和子线程

### 11.7.1 子进程

在Python中，子进程是通过`os.fork()`函数创建的。子进程是父进程的副本，拥有自己的内存空间。子进程可以独立运行，也可以与父进程进行通信。

### 11.7.2 子线程

在Python中，子线程是通过`threading.Thread`类创建的。子线程是父线程的一部分，共享父线程的内存空间。子线程可以独立运行，也可以与父线程进行通信。

```
from multiprocessing.connection import
Listener
import traceback

def
echo_client(conn):
    try
:
        while
True:
            msg = conn.recv()
            conn.send(msg)
    except EOFError
```

```

:
    print
('Connection closed')
def
echo_server(address, authkey):
    serv = Listener(address, authkey=authkey)
    while
True:
    try
:
        client = serv.accept()
        echo_client(client)
    except Exception
:
        traceback.print_exc()
echo_server(('', 25000), authkey=b'peekaboo')

```

□□□□□□□□□□□□□□□□□□□□

```

>>> from multiprocessing.connection import
Client
>>> c = Client(('localhost', 25000), authkey=b'peekaboo')
>>> c.send('hello')
>>> c.recv()
'hello'
>>> c.send(42)
>>> c.recv()
42
>>> c.send([1, 2, 3, 4, 5])
>>> c.recv()
[1, 2, 3, 4, 5]
>>>

```



socket 的 send() 和 recv() 方法使用 pickle 模块来序列化数据。

### 11.7.3 消息队列

ZeroMQ 和 Celery 都使用 socket 来通信。multiprocessing.connection 是一个——primitive——的接口。

UNIX socket 和 Windows 的 UNIX socket 接口。

```
s = Listener('/tmp/myconn', authkey=b'peekaboo')
```

Windows 的 socket 接口。

```
s = Listener(r'\\  
.\pipe\myconn', authkey=b'peekaboo')
```

multiprocessing Client() Listener() authkey

multiprocessing I/O socket

## 11.8

### 11.8.1

socket multiprocessing.connection ZeroMQ RPC

### 11.8.2

pickle  
pickle  
RPC  
RPC

```
# rpcserver.py
```

```
import pickle
```

```
class RPCHandler
```

```
:
```

```
    def
```

```
__init__(self):
```

```
    self._functions = { }
```

```
    def
```

```
register_function(self, func):
```

```
    self._functions[func.__name__] = func
```

```
    def
```

```
handle_connection(self, connection):
```

```
    try
```

```
:
```

```
        while
```

```
True:
```

```
            # Receive a message
```

```
            func_name, args, kwargs =
```

```
pickle.loads(connection.recv())
```

```
            # Run the RPC and send a response
```

```

        try
:
            r = self._functions[func_name]
(*args,**kwargs)
            connection.send(pickle.dumps(r))
        except Exception as
e:
            connection.send(pickle.dumps(e))
        except EOFError
:
            pass

```

multiprocessing
 multiprocessing
 RPC

```

from multiprocessing.connection import
Listener
from threading import
Thread
def
rpc_server(handler, address, authkey):
    sock = Listener(address, authkey=authkey)
    while
True:
        client = sock.accept()
        t = Thread(target=handler.handle_connection, args=
(client,))

```



```

class RPCProxy
:
    def
__init__(self, connection):
    self._connection = connection
    def
__getattr__(self, name):
    def
do_rpc(*args, **kwargs):
    self._connection.send(pickle.dumps((name, args,
kwargs)))
    result = pickle.loads(self._connection.recv())
    if
isinstance(result, Exception
):
        raise
result
    return
result
    return
do_rpc

```

```

████████████████████████████████████████████████████████████████████████████████
██████

```

```

>>> from multiprocessing.connection import
Client
>>> c = Client(('localhost', 17000), authkey=b'peekaboo')
>>> proxy = RPCProxy(c)
>>> proxy.add(2, 3)

```

```

5
>>> proxy.sub(2, 3)
-1
>>> proxy.sub([1, 2], 4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "rpcserver.py", line 37, in do_rpc
    raise

result
TypeError: unsupported operand type(s) for -: 'list' and 'int'
>>>

```

multiprocessing  
 pickle  
 pickle.dumps()  
 pickle.loads()

### 11.8.3

RPCHandler  
 RPCProxy  
 foo(1, 2, z=3)  
 ('foo', (1, 2),  
 {'z':3})  
 pickle  
 RPCProxy  
 \_\_getattr\_\_()  
 do\_rpc()  
 pickle

multiprocessing  
 ZeroMQ

RPC 与 ZeroMQ socket

pickle 序列化与反序列化  
RPC 序列化与反序列化  
RPC 序列化与反序列化  
RPC 序列化与反序列化  
RPC 序列化与反序列化  
RPC 序列化与反序列化

pickle 序列化与反序列化  
JSON 与 XML  
json.loads()  
json.dumps()  
pickle.loads()  
pickle.dumps()  
JSON 序列化与反序列化

```
# jsonrpcserver.py

import json

class RPCHandler:
    def
    __init__(self):
        self._functions = { }
    def
    register_function(self, func):
```



```

        self._functions[func.__name__] = func

    def
handle_connection(self, connection):
    try
:
        while
True:
            # Receive a message

            func_name, args, kwargs =
json.loads(connection.recv())
            # Run the RPC and send a response

            try
:
                r = self._functions[func_name]
(*args,**kwargs)
                connection.send(json.dumps(r))
            except Exception as
e:
                connection.send(json.dumps(str(e)))
            except EOFError
:
                pass

# jsonrpcclient.py

import json

class RPCProxy

```

```

:
    def
__init__(self, connection):
    self._connection = connection
    def
__getattr__(self, name):
    def
do_rpc(*args, **kwargs):
    self._connection.send(json.dumps((name, args,
kwargs)))
    result = json.loads(self._connection.recv())
    return
result
return
do_rpc

```

RPC
 pickle
 JSON

XML-RPC
 SimpleXMLRPCServer
 ServerProxy
 11.6

## 11.9 安全套接字接口

### 11.9.1 简介

安全套接字接口（SSL）是用于在两个应用程序之间建立加密连接的标准接口。它提供了一种在不可靠的网络上实现可靠、安全通信的方法。SSL 协议由 Netscape 公司开发，现已成为互联网上最广泛使用的加密通信协议之一。

### 11.9.2 使用 hmac

hmac 模块提供了一种简单的方法，用于计算和验证消息的摘要。它使用哈希函数（如 SHA-1 或 SHA-256）和密钥来生成摘要。hmac 模块还支持验证摘要，以确保消息在传输过程中没有被篡改。

```
import hmac
```

```
import os
```

```
def
```

```
client_authenticate(connection, secret_key):  
    """
```

```
        Authenticate client to a remote service.
```

```
        connection represents a network connection.
```

```
        secret_key is a key known only to both client/server.
```

```

'''

message = connection.recv(32)
hash = hmac.new(secret_key, message)
digest = hash.digest()
connection.send(digest)

def
server_authenticate(connection, secret_key):
'''

    Request client authentication.

'''

message = os.urandom(32)
connection.send(message)
hash = hmac.new(secret_key, message)
digest = hash.digest()
response = connection.recv(len(digest))
return

hmac.compare_digest(digest, response)

```

```

    os.urandom()
    hmac
    hash
    digest

```

```

    hmac.compare_digest()
    timing-

```

analysis attack  
==

socket

```
from socket import
socket, AF_INET, SOCK_STREAM

secret_key = b'peekaboo'
def
echo_handler(client_sock):
    if not
server_authenticate(client_sock, secret_key):
    client_sock.close()
    return

while
True:
    msg = client_sock.recv(8192)
    if not
msg:
        break

    client_sock.sendall(msg)

def
echo_server(address):
    s = socket(AF_INET, SOCK_STREAM)
    s.bind(address)
    s.listen(5)
    while
```

```
echo_server(('', 18000))
```

```
s = socket(AF_INET, SOCK_STREAM)
s.connect(('localhost', 18000))
client_authenticate(s, secret_key)
s.send(b'Hello World')
resp = s.recv(1024)
...
```

```

    hmac
multiprocessing
hmac

```

hmac 模块使用 HMAC 消息认证码，使用 MD5 或 SHA-1 作为哈希函数。IETF RFC 2104  
<http://tools.ietf.org/html/rfc2104.html>

hmac 模块使用 HMAC 消息认证码，使用 MD5 或 SHA-1 作为哈希函数。IETF RFC 2104  
<http://tools.ietf.org/html/rfc2104.html>

## 11.10 SSL 模块

### 11.10.1 简介

SSL 模块使用 socket 模块提供的套接字，并支持 SSL 协议。

### 11.10.2 使用

ssl 模块使用 socket 模块提供的套接字，并支持 SSL 协议。使用 ssl.wrap\_socket() 方法将 socket 对象包装成 SSL 对象。echo 方法用于回显数据。

```
from socket import  
socket, AF_INET, SOCK_STREAM
```

```

import ssl

KEYFILE = 'server_key.pem'           # Private key of the server

CERTFILE = 'server_cert.pem'        # Server certificate (given
to client)

def
echo_client(s):
    while
True:
        data = s.recv(8192)
        if
data == b'':
            break

        s.send(data)
        s.close()
        print
('Connection closed')

def
echo_server(address):
    s = socket(AF_INET, SOCK_STREAM)
    s.bind(address)
    s.listen(1)

    # Wrap with an SSL layer requiring client certs

    s_ssl = ssl.wrap_socket(s,
                             keyfile=KEYFILE,
                             certfile=CERTFILE,
                             server_side=True

```





```
ca_certs = 'server_cert.pem')
>>> s_ssl.connect(('localhost', 20000))
>>> s_ssl.send(b'Hello World?')
12
>>> s_ssl.recv(8192)
b'Hello World?'
>>>
```

socket server 模块  
HTTP XML-RPC socketserver 模块  
SSL 模块

mixins 类  
SSL 类

```
import ssl

class SSLMixin
:
    ...

    Mixin class that adds support for SSL to existing servers
    based

    on the socketserver module.

    ...
```



```
SimpleXMLRPCServer
```

```
class SSLSimpleXMLRPCServer
```

```
(SSLMixin, SimpleXMLRPCServer):  
    pass
```

Python XML-RPC 라이브러리 11.6 버전부터 SSL  
SSL

```
import ssl
```

```
from xmlrpc.server import
```

```
SimpleXMLRPCServer
```

```
from sslmixin import
```

```
SSLMixin
```

```
class SSLSimpleXMLRPCServer
```

```
(SSLMixin, SimpleXMLRPCServer):  
    pass
```

```
class KeyValueServer
```

```
:
```

```
    _rpc_methods_ = ['get', 'set', 'delete', 'exists', 'keys']  
    def
```

```
    __init__(self, *args, **kwargs):
```

```
        self._data = {}
```

```
        self._serv = SSLSimpleXMLRPCServer(*args,
```

```

allow_none=True, **kwargs)
    for
name in
self._rpc_methods_:
    self._serv.register_function(getattr(self, name))

    def
get(self, name):
    return
self._data[name]

    def
set(self, name, value):
    self._data[name] = value

    def
delete(self, name):
    del
self._data[name]

    def
exists(self, name):
    return
name in
self._data

    def
keys(self):
    return
list(self._data)

    def

```

```

serve_forever(self):
    self._serv.serve_forever()

if
__name__ == '__main__':
    KEYFILE='server_key.pem'      # Private key of the server

    CERTFILE='server_cert.pem'    # Server certificate

    kvserv = KeyValueServer('', 15000),
                                keyfile=KEYFILE,
                                certfile=CERTFILE)
    kvserv.serve_forever()

```

xmlrpc.client
 URL
 https:

```

>>> from xmlrpc.client import
ServerProxy
>>> s = ServerProxy('https://localhost:15000',
allow_none=True)
>>> s.set('foo','bar')
>>> s.set('spam', [1, 2, 3])
>>> s.keys()
['spam', 'foo']
>>> s.get('foo')
'bar'
>>> s.get('spam')
[1, 2, 3]
>>> s.delete('spam')
>>> s.exists('spam')
False
>>>

```

SSL  
XML-RPC

```
from xmlrpc.client import
SafeTransport, ServerProxy
import ssl

class VerifyCertSafeTransport
(SafeTransport):
    def
__init__(self, cafile, certfile=None, keyfile=None):
    SafeTransport.__init__(self)
    self._ssl_context = ssl.SSLContext(ssl.PROTOCOL_TLSv1)
    self._ssl_context.load_verify_locations(cafile)
    if
cert:
        self._ssl_context.load_cert_chain(certfile,
keyfile)
        self._ssl_context.verify_mode = ssl.CERT_REQUIRED

    def
make_connection(self, host):
    # Items in the passed dictionary are passed as keyword

    # arguments to the http.client.HTTPSConnection()
constructor.

    # The context argument allows an ssl.SSLContext
instance to
```

```
# be passed with information about the SSL
configuration
```

```
s = super().make_connection((host, {'context':
self._ssl_context}))
```

**return**

**S**

```
# Create the client proxy
```

```
s = ServerProxy('https://localhost:15000',
transport=VerifyCertSafeTransport('server_cert.pem'),
allow_none=True)
```

**if**

```
__name__ == '__main__':  
    KEYFILE='server_key.pem'           # Private key of the server
```

```
CERTFILE='server cert.pem'    # Server certificate
```

```
CA_CERTS='client_cert.pem'      # Certificates of accepted
clients
```



```

kvserv = KeyValueServer('', 15000),
                        keyfile=KEYFILE,
                        certfile=CERTFILE,
                        ca_certs=CA_CERTS,
                        cert_reqs=ssl.CERT_REQUIRED,
                        )
kvserv.serve_forever()

```

## XML-RPC ServerProxy

```

# Create the client proxy

s = ServerProxy('https://localhost:15000',
transport=VerifyCertSafeTransport('server_cert.pem',
'client_cert.pem',
'client_key.pem'),
allow_none=True)

```

### 11.10.3

SSL

SSL  
Verisign  
Equifax  
Web  
HTTPS  
Web

```
bash % openssl req -new -x509 -days 365 -nodes -out  
server_cert.pem \
```

```
        -keyout server_key.pem  
Generating a 1024 bit RSA private key  
.....++++++  
...++++++  
writing new private key to 'server_key.pem'  
-----  
You are about to be asked to enter information that will be  
incorporated  
into your certificate request.  
What you are about to enter is what is called a Distinguished  
Name or a DN.  
There are quite a few fields but you can leave some blank  
For some fields there will be a default value,  
If you enter '.', the field will be left blank.  
-----  
Country Name (2 letter code) [AU]:US  
State or Province Name (full name) [Some-State]:Illinois  
Locality Name (eg, city) []:Chicago  
Organization Name (eg, company) [Internet Widgits Pty  
Ltd]:Dabeaz, LLC
```



```
YbTvqKc7AkEAwnRB02VYEZsJZp2X0IZqP9ovWokkpYx+PE4+c6MySDgaMcigL
7v
WDIHJG1CHudD09GbqENasDzyb2HAIW4CzQJBAKDdkv+xoW6gJx42Auc2WzTcUH
CA
eXR/+BLpPrhKykbv0Q8YvS5W764SU01u1LWs3G+wnRMvrRv1MCZKgggBjkCQO
CG
Jewto2+a+Wk0KQXrNNScCDE5aPTmZQc5waCYq4UmCZQc0jkU0iN3ST1U5iuxRq
fb
V/yX6fw0qh+fLWtk0s/JAkA+okMSxZwqRtfg0FGBfwQ8/iKrnizeanTQ3L6scF
XI
CHZXdJ3XQ6qUmNxNn7iJ7S/LDawo1QfWkCfD9FYoxBlg
-----END RSA PRIVATE KEY-----
```

server\_cert.pem

```
-----BEGIN CERTIFICATE-----
MIIC+DCCAmGgAwIBAgIJAPMd+vi45js3MA0GCSqGSIb3DQEBBQUAMFwxCzAJBg
NV
BAYTA1VTMREwDwYDVQQIEWhJbGxpbn9pczEQMA4GA1UEBxMHQ2hpY2FnbzEUMB
IG
A1UEChMLRGFiZWV6LCBMTEMxEjAQBgNVBAMTCWxvY2FsaG9zdDAeFw0xMzAxMT
Ex
ODQyMjdaFw0xNDExMTEwODQyMjdaMFwxCzAJBgNVBAYTA1VTMREwDwYDVQQIEw
hJ
bGxpbn9pczEQMA4GA1UEBxMHQ2hpY2FnbzEUMBIGA1UEChMLRGFiZWV6LCBMTE
Mx
EjAQBgNVBAMTCWxvY2FsaG9zdDCBnzANBgkqhkiG9w0BAQEFAA0BjQAwgYkCgY
EA
mawjS6BMgChfn/VDXBWs+TrGuo3+6pG1JfLIucUK2N2WAu47rpy9XWS5/1WxBS
CE
```

2lDoLwbT79aIFkyRsIGutlUhtaBRNDgyMd4NjYeLEX/q8krMdi+00Np8dM+Dub  
yU

050nkTRwGVFJwi+dPmL48i8re68i0o0rioQnCbG2YD8CAwEAAa0BwTCBvjAdBg  
NV

HQ4EFgQUrtoLHHgXiDZTr26NMmgKJLJLFtIwgY4GA1UdIwSBhjCBg4AUrtoLHH  
gX

iDZTr26NMmgKJLJLFtKhYKReMFwxCzAJBgNVBAYTA1VTMREwDwYDVQQIEwhJbG  
xp

bm9pczEQMA4GA1UEBxMHQ2hpY2FnbzEUMBGA1UEChMLRGFiZWY6LCBMTEMxEj  
AQ

BgNVBAMTCWxvY2FsaG9zdIIJAPMd+vi45js3MAwGA1UdEwQFMAMBAf8wDQYJKo  
ZI

hvcNAQEFBQADgYEAFCi+dqvMG4xF8UTnbGVvZJPIzJDRee6Nbt6AHQo9p0dAIM  
Au

WsGCplS0aDNdKKzl+b2UT2Zp3AIW4Qd51bouSNnR4M/gnr9ZD1ZctFd3jS+C5X  
Rp

D3vvcW5lAnCCC80P6rXy7d7hTeFu5EYKtRGXNvVNd/06NALGDflrr0wxF3Y=  
-----END CERTIFICATE-----

SSL  
-----

-----

Python 3.6.0 SSL 라이브러리  
 Python 3.6.0 SSL 라이브러리  
 http://docs.python.org/3/library/ssl.html  
 Python 3.6.0 SSL 라이브러리  
 Python 3.6.0

## 11.11 socket

### 11.11.1 ☐ ☐

Python

### 11.11.2 □□□□

UNIX  
UNIX socket Windows  
multiprocessing

multiprocessing.reduction  
send\_handle()recv\_handle()

```
import multiprocessing

from multiprocessing.reduction import
recv_handle, send_handle
import socket

def
worker(in_p, out_p):
    out_p.close()
    while
True:
        fd = recv_handle(in_p)
        print
('CHILD: GOT FD', fd)
        with
socket.socket(socket.AF_INET, socket.SOCK_STREAM, fileno=fd)
as
s:
        while
True:
            msg = s.recv(1024)
            if not
msg:
                break
```

```

        print

('CHILD: RECV {!r}'.format(msg))
        s.send(msg)

def
server(address, in_p, out_p, worker_pid):
    in_p.close()
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)
    s.bind(address)
    s.listen(1)
    while
True:
        client, addr = s.accept()
        print

('SERVER: Got connection from', addr)
        send_handle(out_p, client.fileno(), worker_pid)
        client.close()

if
__name__ == '__main__':
    c1, c2 = multiprocessing.Pipe()
    worker_p = multiprocessing.Process(target=worker, args=
(c1,c2))
    worker_p.start()

    server_p = multiprocessing.Process(target=server,
        args=('', 15000), c1, c2, worker_p.pid))
    server_p.start()

    c1.close()
    c2.close()

```



multiprocessing.Pipe  
socket.recv\_handle()  
socket.send\_handle()  
socket

Telnet

```
bash % python3 passfd.py
SERVER: Got connection from ('127.0.0.1', 55543)
CHILD: GOT FD 7
CHILD: RECV b'Hello\r\n'
CHILD: RECV b'World\r\n'
```

socket

### 11.11.3

Python

send\_handle()recv\_handle()  
11.7  
UNIXsocket  
Windows  
socket

```
# servermp.py
```

```
from multiprocessing.connection import
```

```
Listener
```

```
from multiprocessing.reduction import
```

```
send_handle
```

```
import socket
```

```
def
```

```
server(work_address, port):
```

```
    # Wait for the worker to connect
```

```
    work_serv = Listener(work_address, authkey=b'peekaboo')
```

```
    worker = work_serv.accept()
```

```
    worker_pid = worker.recv()
```

```
    # Now run a TCP/IP server and send clients to worker
```

```
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)
```

```
    s.bind(('', port))
```

```
    s.listen(1)
```

```
    while
```

```
True:
```

```

        client, addr = s.accept()
        print
('SERVER: Got connection from', addr)
        send_handle(worker, client.fileno(), worker_pid)
        client.close()

if
__name__ == '__main__':
    import sys

    if
len(sys.argv) != 3:
        print
('Usage: server.py server_address port', file=sys.stderr)
        raise SystemExit

(1)

    server(sys.argv[1], int(sys.argv[2]))

```

python3  
servermp.py /tmp/servconn 15000

```

# workermp.py

from multiprocessing.connection import
Client
from multiprocessing.reduction import

```

```

recv_handle
import os

from socket import

socket, AF_INET, SOCK_STREAM

def

worker(server_address):
    serv = Client(server_address, authkey=b'peekaboo')
    serv.send(os.getpid())
    while

True:
    fd = recv_handle(serv)
    print

('WORKER: GOT FD', fd)
    with

socket(AF_INET, SOCK_STREAM, fileno=fd) as

client:
    while

True:
        msg = client.recv(1024)
        if not

msg:
            break

            print

('WORKER: RECV {!r}'.format(msg))
            client.send(msg)

if

__name__ == '__main__':
    import sys

```

```

    if
len(sys.argv) != 2:
    print
('Usage: worker.py server_address', file=sys.stderr)
    raise SystemExit
(1)
worker(sys.argv[1])

```

python3 workerm.py  
/tmp/servconnPipe()

UNIX socket  
socket.sendmsg()  
socket

*# server.py*

**import socket**

**import struct**

**def**

send\_fd(sock, fd):  
 , , ,

```

    Send a single file descriptor.

    '''

    sock.sendmsg([b'x'],
                  [(socket.SOL_SOCKET, socket.SCM_RIGHTS,
struct.pack('i', fd))])
    ack = sock.recv(2)
    assert

ack == b'OK'

def

server(work_address, port):
    # Wait for the worker to connect

    work_serv = socket.socket(socket.AF_UNIX,
socket.SOCK_STREAM)
    work_serv.bind(work_address)
    work_serv.listen(1)
    worker, addr = work_serv.accept()

    # Now run a TCP/IP server and send clients to worker

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)
    s.bind(('',port))
    s.listen(1)
    while

True:
    client, addr = s.accept()
    print

('SERVER: Got connection from', addr)
    send_fd(worker, client.fileno())
    client.close()

```

```

if
__name__ == '__main__':
    import sys

    if
len(sys.argv) != 3:
    print
('Usage: server.py server_address port', file=sys.stderr)
    raise SystemExit

(1)

    server(sys.argv[1], int(sys.argv[2]))

```

□□□□socket□□□□□□□□□□

```

# worker.py

import socket

import struct

def
recv_fd(sock):
    '''

    Receive a single file descriptor

    '''

```

```

    msg, ancdata, flags, addr = sock.recvmsg(1,
socket.CMSG_LEN(struct.calcsize('i')))

    cmsg_level, cmsg_type, cmsg_data = ancdata[0]
    assert

cmsg_level == socket.SOL_SOCKET and

cmsg_type == socket.SCM_RIGHTS
    sock.sendall(b'OK')
    return

struct.unpack('i', cmsg_data)[0]

def

worker(server_address):
    serv = socket.socket(socket.AF_UNIX, socket.SOCK_STREAM)
    serv.connect(server_address)
    while

True:
    fd = recv_fd(serv)
    print

('WORKER: GOT FD', fd)
    with

socket.socket(socket.AF_INET, socket.SOCK_STREAM, fileno=fd)
as

client:
    while

True:
        msg = client.recv(1024)
        if not

msg:
        break

```





## 11.12.1 `select`

`select` is a module in Python that implements “select” and “poll” I/O multiplexing. It is a low-level interface to the operating system’s `select` system call. It is used to wait for multiple file descriptors to become ready for I/O.

## 11.12.2 `selectors`

`selectors` is a module in Python that implements I/O multiplexing. It is a high-level interface to the operating system’s `select` system call. It is used to wait for multiple file descriptors to become ready for I/O. `socket` is a module in Python that implements “socket” and “poll” I/O multiplexing. It is a high-level interface to the operating system’s `select` system call. It is used to wait for multiple file descriptors to become ready for I/O.

```
class EventHandler
:
    def
fileno(self):
    'Return the associated file descriptor'
    raise
NotImplemented('must implement')
    def
wants_to_receive(self):
    'Return True if receiving is allowed'
    return
False
```

```
def
handle_receive(self):
    'Perform the receive operation'
    pass

def
wants_to_send(self):
    'Return True if sending is requested'
    return
False

def
handle_send(self):
    'Send outgoing data'
    pass
```

[illegible]

```
import select

def
event_loop(handlers):
    while
True:
        wants_recv = [h for
h in
```

```

handlers if
h.wants_to_receive()]
    wants_send = [h for
h in
handlers if
h.wants_to_send()]
    can_recv, can_send, _ = select.select(wants_recv,
wants_send, [])
    for
h in
can_recv:
    h.handle_receive()
for
h in
can_send:
    h.handle_send()

```

select() select() select()  
 select() select()  
 select() select()  
 select() select()  
 handle\_receive()  
 handle\_send()

EventHandler  
 UDP

```
import socket
```

```
import time
```

```
class UDPServer
```

```
(EventHandler):
```

```
    def
```

```
    __init__(self, address):
```

```
        self.sock = socket.socket(socket.AF_INET,  
socket.SOCK_DGRAM)
```

```
        self.sock.bind(address)
```

```
    def
```

```
    fileno(self):
```

```
        return
```

```
self.sock.fileno()
```

```
    def
```

```
    wants_to_receive(self):
```

```
        return
```

```
True
```

```
class UDPTimeServer
```

```
(UDPServer):
```

```
    def
```

```
    handle_receive(self):
```

```
        msg, addr = self.sock.recvfrom(1)
```

```
        self.sock.sendto(time.ctime().encode('ascii'), addr)
```

```
class UDPEchoServer
```

```
(UDPServer):
```

```
    def
```

```

handle_receive(self):
    msg, addr = self.sock.recvfrom(8192)
    self.sock.sendto(msg, addr)

if
__name__ == '__main__':
    handlers = [ UDPTimerServer('',14000)),
UDPEchoServer('',15000)) ]
    event_loop(handlers)

```

## Python

```

>>> from socket import
*
>>> s = socket(AF_INET, SOCK_DGRAM)
>>> s.sendto(b'',('localhost',14000))
0
>>> s.recvfrom(128)
(b'Tue Sep 18 14:29:23 2012', ('127.0.0.1', 14000))
>>> s.sendto(b'Hello',('localhost',15000))
5
>>> s.recvfrom(128)
(b'Hello', ('127.0.0.1', 15000))
>>>

```

## TCPTCP echo

```

class TCPServer
(EventHandler):
    def

```

```

__init__(self, address, client_handler, handler_list):
    self.sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    self.sock.setsockopt(socket.SOL_SOCKET,
socket.SO_REUSEADDR, True)
    self.sock.bind(address)
    self.sock.listen(1)
    self.client_handler = client_handler
    self.handler_list = handler_list

    def

fileno(self):
    return

self.sock.fileno()

    def

wants_to_receive(self):
    return

True

    def

handle_receive(self):
    client, addr = self.sock.accept()
    # Add the client to the event loop's handler list

    self.handler_list.append(self.client_handler(client,
self.handler_list))
class TCPClient
(EventHandler):
    def

__init__(self, sock, handler_list):
    self.sock = sock
    self.handler_list = handler_list
    self.outgoing = bytearray()

    def

```

```

fileno(self):
    return

self.sock.fileno()

    def

close(self):
    self.sock.close()
    # Remove myself from the event loop's handler list

    self.handler_list.remove(self)

    def

wants_to_send(self):
    return

True if

self.outgoing else

False

    def

handle_send(self):
    nsent = self.sock.send(self.outgoing)
    self.outgoing = self.outgoing[nsent:]

class TCPEchoClient

(TCPClient):
    def

wants_to_receive(self):
    return

True

    def

handle_receive(self):

```



```

        data = self.sock.recv(8192)
        if not
data:
            self.close()
        else
:
            self.outgoing.extend(data)
if
__name__ == '__main__':
    handlers = []
    handlers.append(TCPServer(('',16000), TCPEchoClient,
handlers))
    event_loop(handlers)

```

TCP

Telnet

## 11.12.3

socket

非阻塞I/O编程模型  
使用select()函数  
socket编程模型  
阻塞I/O编程模型

非阻塞I/O编程模型  
使用select()函数  
socket编程模型  
阻塞I/O编程模型

非阻塞I/O编程模型  
使用select()函数  
socket编程模型  
concurrent.futures

```
from concurrent.futures import  
ThreadPoolExecutor  
import os  
  
class ThreadPoolHandler  
(EventHandler):  
    def  
__init__(self, nworkers):  
    if  
os.name == 'posix':  
        self.signal_done_sock, self.done_sock =  
socket.socketpair()  
    else
```

```

:
    server = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    server.bind(('127.0.0.1', 0))
    server.listen(1)
    self.signal_done_sock =
socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
self.signal_done_sock.connect(server.getsockname())
    self.done_sock, _ = server.accept()
    server.close()

    self.pending = []
    self.pool = ThreadPoolExecutor(nworkers)

    def
fileno(self):
        return
self.done_sock.fileno()

    # Callback that executes when the thread is done

    def
_complete(self, callback, r):
    self.pending.append((callback, r.result()))
    self.signal_done_sock.send(b'x')

    # Run a function in a thread pool

    def
run(self, func, args=(), kwargs={}, *, callback):
    r = self.pool.submit(func, *args, **kwargs)
    r.add_done_callback(lambda
r: self._complete(callback, r))

    def

```

```

wants_to_receive(self):
    return

True

    # Run callback functions of completed work

    def

handle_receive(self):
    # Invoke all pending callback functions

    for

callback, result in

self.pending:
    callback(result)
    self.done_sock.recv(1)
    self.pending = []

```

```

    def run():
        # ...
    ThreadPoolExecutor(
        # ...
    socket
    _complete()
    socket
    fileno()
    socket
    handle_receive()
    # ...

```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□

```
# A really bad Fibonacci implementation

def
fib(n):
    if
n < 2:
        return
1
    else
:
        return
fib(n - 1) + fib(n - 2)

class UDPFibServer
(UDPServer):
    def
handle_receive(self):
        msg, addr = self.sock.recvfrom(128)
        n = int(msg)
        pool.run(fib, (n,), callback=lambda
r: self.respond(r, addr))

    def
respond(self, result, addr):
        self.sock.sendto(str(result).encode('ascii'), addr)
if
__name__ == '__main__':
    pool = ThreadPoolHandler(16)
```

```
handlers = [ pool, UDPFibServer('',16000)]  
event_loop(handlers)
```

Python  
Python

```
from socket import  
  
*  
sock = socket(AF_INET, SOCK_DGRAM)  
for  
x in  
range(40):  
    sock.sendto(str(x).encode('ascii'), ('localhost', 16000))  
    resp = sock.recvfrom(8192)  
    print  
(resp[0])
```

Python

Python  
Python  
Python  
Python  
12.12

## 11.13 内存映射

### 11.13.1 内存映射

内存映射是一种将文件内容映射到内存的技术，使得程序可以直接通过内存访问文件内容，而不需要通过文件系统。这通常用于处理大型文件或需要快速访问的数据。

### 11.13.2 内存映射

内存映射通常使用 `memoryview` 对象来实现，它提供了一种高效的方式来访问内存数据。

```
# zerocopy.py

def
send_from(arr, dest):
    view = memoryview(arr).cast('B')
    while
len(view):
    nsent = dest.send(view)
    view = view[nsent:]

def
recv_into(arr, source):
    view = memoryview(arr).cast('B')
    while
len(view):
    nrecv = source.recv_into(view)
    view = view[nrecv:]
```

socket

```
>>> from socket import *
>>> s = socket(AF_INET, SOCK_STREAM)
>>> s.bind(('', 25000))
>>> s.listen(1)
>>> c,a = s.accept()
>>>
```

```
>>> from socket import *
>>> c = socket(AF_INET, SOCK_STREAM)
>>> c.connect(('localhost', 25000))
>>>
```

array numpy

```
# Server
>>> import numpy

>>> a = numpy.arange(0.0, 50000000.0)
>>> send_from(a, c)
```



```

>>>

# Client
>>> import numpy

>>> a = numpy.zeros(shape=50000000, dtype=float)
>>> a[0:10]
array([ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
>>> recv_into(a, c)
>>> a[0:10]
array([ 0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])
>>>

```

## 11.13.3 内存池

内存池是一种内存管理技术，它通过预先分配一块内存，并将其划分为多个小块，以便在需要时快速分配和释放。这种技术可以减少内存碎片化，提高内存利用率。在Python中，内存池通常用于管理大量的小对象，如字符串、列表等。通过内存池，可以显著减少垃圾回收的频率，从而提高程序的性能。

内存池的实现通常涉及到对内存的预先分配和划分。在Python中，内存池可以通过`memoryview`和`memoryview`模块来实现。这些模块允许用户直接访问和操作内存池中的小块内存，从而避免了频繁的垃圾回收操作。此外，内存池还可以用于管理大量的字符串对象，通过共享内存来减少内存占用。

内存池的使用可以显著提高程序的性能，特别是在处理大量小对象的情况下。通过预先分配内存，可以避免频繁的内存分配和释放操作，从而减少垃圾回收的频率。此外，内存池还可以用于管理大量的字符串对象，通过共享内存来减少内存占用。在Python中，内存池可以通过`memoryview`和`memoryview`模块来实现。这些模块允许用户直接访问和操作内存池中的小块内存，从而避免了频繁的垃圾回收操作。

memoryview 是 Python 3 中引入的一个新特性，它提供了一种高效且简单的方式，可以查看任意对象的内存内容，而无需知道底层的内存布局。

```
view = memoryview(arr).cast('B')
```

arr 是一个 NumPy 数组，memoryview 是 Python 3 中引入的一个新特性，它提供了一种高效且简单的方式，可以查看任意对象的内存内容，而无需知道底层的内存布局。

memoryview 对象可以用于与 socket 交互。通过 sock.send() 和 sock.recv\_into() 方法，可以将 memoryview 对象的数据发送到 socket 或从 socket 接收数据。memoryview 对象还支持切片操作，可以用于处理数据子集。

memoryview 对象可以用于与 socket 交互。通过 sock.send() 和 sock.recv\_into() 方法，可以将 memoryview 对象的数据发送到 socket 或从 socket 接收数据。memoryview 对象还支持切片操作，可以用于处理数据子集。

memoryview 对象可以用于与 socket 交互。通过 sock.send() 和 sock.recv\_into() 方法，可以将 memoryview 对象的数据发送到 socket 或从 socket 接收数据。memoryview 对象还支持切片操作，可以用于处理数据子集。



## 12 线程

Python 多线程编程，是指同时执行多个子程序或线程的过程。多线程编程主要应用在需要大量同时运行时的工作场合，通过多线程编程实现程序同时执行若干个子程序，从而提高程序的运行效率。在 Windows 操作系统中，多线程编程可以通过调用 Windows API 函数来实现。

多线程编程的主要优点是能够充分利用处理器的空闲时间，提高程序的运行效率。同时，多线程编程还可以实现程序的并行执行，从而缩短程序的运行时间。然而，多线程编程也存在一些缺点，例如线程之间的同步问题、死锁问题等。

### 12.1 多线程编程

#### 12.1.1 线程

线程是程序中的一个执行单元，它能够独立地执行一段程序。在多线程编程中，一个程序可以包含多个线程，每个线程都可以独立地执行自己的任务。

#### 12.1.2 线程库

Python 提供了 `threading` 模块，用于实现多线程编程。该模块包含 `Thread` 类，用于创建和管理线程。通过 `Thread` 类，可以轻松地创建新的线程，并将需要执行的代码放入线程中。

```
# Code to execute in an independent thread
```

```
import time
```

```

def
countdown(n):
    while
n > 0:
        print
('T-minus', n)

n -= 1

time.sleep(5)

# Create and launch a thread

from threading import
Thread
t = Thread(target=countdown, args=(10,))
t.start()

```

start()

POSIX
 Windows

```

if

```

```
t.is_alive():
    print
('Still running')
else
:
    print
('Completed')
```

join()은 스레드가 종료될 때까지 기다리는 방법입니다.

```
t.join()
```

daemon 스레드는 프로그램이 종료될 때 자동으로 종료됩니다.  
daemon=True로 설정하면 스레드가 데몬 스레드가 됩니다.  
daemon은 스레드의 속성입니다.

```
t = Thread(target=countdown, args=(10,), daemon=True)
t.start()
```

daemon 스레드는 프로그램이 종료될 때 자동으로 종료됩니다.  
daemon=True로 설정하면 스레드가 데몬 스레드가 됩니다.  
daemon은 스레드의 속성입니다.

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
class CountdownTask
:
    def
__init__(self):

self._running = True


    def
terminate(self):

self._running = False


    def
run(self, n):
        while
self._running and
n > 0:
            print
('T-minus', n)
```

```

n -= 1

time.sleep(5)

c = CountdownTask()
t = Thread(target=c.run, args=(10,))
t.start()
...
c.terminate()      # Signal termination

t.join()           # Wait for actual termination (if needed)

```

```

      I/O
I/O
I/O
I/O

```

```

class IOTask
:
    def
terminate(self):

self._running = False

    def
run(self, sock):

# sock is a socket

```



```
sock.settimeout(5)           # Set timeout period

    while
self._running:

# Perform a blocking I/O operation w/ timeout

        try

:

data = sock.recv(8192)
        break

        except

socket.timeout:
        continue

# Continued processing

...

# Terminated

    return
```

## 12.1.3 线程

Python 解释器使用 GIL 来保证 Python 解释器在任意时刻只执行一个线程。Python 解释器使用 GIL 来保证 Python 解释器在任意时刻只执行一个线程。Python 解释器使用 GIL 来保证 Python 解释器在任意时刻只执行一个线程。CPU 密集型任务 Python 解释器 I/O 密集型任务 Python 解释器 I/O 密集型任务 Python 解释器 I/O 密集型任务

Thread 类

```
from threading import
Thread
class CountdownThread
(Thread):
    def
__init__(self, n):
    super().__init__()
    self.n = 0
    def
run(self):
        while
self.n > 0:
```

```
        print
('T-minus', self.n)

self.n -= 1

time.sleep(5)

c = CountdownThread(5)
c.start()
```

threading multiprocessing

```
import multiprocessing

c = CountdownTask(5)
p = multiprocessing.Process(target=c.run)
p.start()
...
```

CountdownTask

## 12.2 线程局部存储

### 12.2.1 简介

线程局部存储（Thread-Local Storage, TLS）是一种存储模型，用于在多线程环境中为每个线程提供独立的变量副本。

### 12.2.2 使用

在 C 语言中，使用 `_Thread_local` 关键字来声明线程局部变量。在 C++ 中，使用 `thread_local` 关键字。在 Python 中，使用 `threading.local()` 来创建线程局部对象。在 Java 中，使用 `ThreadLocal` 类。在 JavaScript 中，使用 `ThreadLocal` 类。在 Go 中，使用 `context.WithValue` 来创建线程局部上下文。

在 Python 中，使用 `threading.local()` 来创建线程局部对象。在 C++ 中，使用 `thread_local` 来声明线程局部变量。在 C 语言中，使用 `_Thread_local` 来声明线程局部变量。在 Java 中，使用 `ThreadLocal` 类来创建线程局部对象。在 JavaScript 中，使用 `ThreadLocal` 类来创建线程局部对象。在 Go 中，使用 `context.WithValue` 来创建线程局部上下文。

在 Python 中，使用 `threading.local()` 来创建线程局部对象。

```
from threading import
```

```
Thread, Event
import time
```

```
# Code to execute in an independent thread

def
countdown(n, started_evt):
    print
    ('countdown starting')

    started_evt.set()
    while
n > 0:
        print
        ('T-minus', n)

    n -= 1

    time.sleep(5)

# Create the event object that will be used to signal startup

started_evt = Event()

# Launch the thread and pass the startup event

print

('Launching countdown')
t = Thread(target=countdown, args=(10,started_evt))
t.start()

# Wait for the thread to start
```

```
started_evt.wait()  
print  
  
('countdown is running')
```

```

    cout<<"countdown is
running"<<"countdown starting"<<endl
    countdown()

```

### 12.2.3 ☐ ☐

[illegible]

Condition

```
import threading

import time
```

```
class PeriodicTimer
:
    def
__init__(self, interval):

self._interval = interval

self._flag = 0

self._cv = threading.Condition()

    def
start(self):

t = threading.Thread(target=self.run)

t.daemon = True

t.start()

    def
run(self):

'''
```

*Run the timer and notify waiting threads after each interval*

```

'''

        while
True:

time.sleep(self._interval)
        with

self._cv:

self._flag ^= 1

self._cv.notify_all()
def
wait_for_tick(self):

'''

Wait for the next tick of the timer

'''

        with

self._cv:

last_flag = self._flag
        while

```



```

last_flag == self._flag:

self._cv.wait()

# Example use of the timer

ptimer = PeriodicTimer(5)
ptimer.start()

# Two threads that synchronize on the timer

def
countdown(nticks):
    while
nticks > 0:

ptimer.wait_for_tick()
    print
('T-minus', nticks)

nticks -= 1

def
countup(last):

n = 0
    while
n < last:

ptimer.wait_for_tick()
    print

```

```

('Counting', n)

n += 1

threading.Thread(target=countdown, args=(10,)).start()
threading.Thread(target=countup, args=(5,)).start()

```

Event Semaphore Condition

semaphore

*# Worker thread*

**def**

worker(n, sema):

*# Wait to be signaled*

sema.acquire()

*# Do some work*

**print**

('Working', n)

```
sema = threading.Semaphore(0)
nworkers = 10
for
n in
range(nworkers):
t = threading.Thread(target=worker, args=(n, sema,))
t.start()
```

```
>>> sema.release()
Working 0
>>> sema.release()
Working 1
>>>
```

[illegible]

## 12.3

## 12.3.1

## 12.3.2

```
from queue import
Queue
from threading import
Thread

# A thread that produces data

def
producer(out_q):
    while
    True:

# Produce some data
```

```
...

out_q.put(data)

# A thread that consumes data

    def
consumer(in_q):
    while
True:

    # Get some data

data = in_q.get()

    # Process the data

...

# Create the shared queue and launch both threads

q = Queue()
t1 = Thread(target=consumer, args=(q,))
t2 = Thread(target=producer, args=(q,))
t1.start()
t2.start()
```

Queue  
producer  
consumer

producer  
consumer

```
from queue import
Queue
from threading import
Thread

# Object that signals shutdown

_sentinel = object()

# A thread that produces data

def
producer(out_q):
    while
    running:

# Produce some data

...
```

```
out_q.put(data)
```

```
# Put the sentinel on the queue to indicate completion
```

```
out_q.put(_sentinel)
```

```
# A thread that consumes data
```

```
def
```

```
consumer(in_q):
```

```
    while
```

```
True:
```

```
# Get some data
```

```
data = in_q.get()
```

```
# Check for termination
```

```
    if
```

```
data is
```

```
_sentinel:
```

```
in_q.put(_sentinel)
```

```
    break
```





```

self._count = 0

self._cv = threading.Condition()
    def
put(self, item, priority):
    with

self._cv:

heapq.heappush(self._queue, (-priority, self._count, item))

self._count += 1

self._cv.notify()
def
get(self):
    with

self._cv:
    while

len(self._queue) == 0:

self._cv.wait()
    return

heapq.heappop(self._queue)[-1]

```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
 □□Queue□□□□□□□□□□□□□□□□□□□□□□□□□□□□completion

```
feature = task_done().join()
```

```
from queue import
Queue
from threading import
Thread

# A thread that produces data

def
producer(out_q):
    while
running:

# Produce some data

...

out_q.put(data)

# A thread that consumes data

def
consumer(in_q):
    while
True:
```

```
# Get some data
```

```
data = in_q.get()
```

```
# Process the data
```

■ ■ ■

```
# Indicate completion
```

```
in_q.task_done()
```

```
# Create the shared queue and launch both threads
```

```
q = Queue()
t1 = Thread(target=consumer, args=(q,))
t2 = Thread(target=producer, args=(q,))
t1.start()
t2.start()
```

```
# Wait for all produced items to be consumed
```

```
q.join()
```

**Event**

```
from queue import
```

```
Queue
```

```
from threading import
```

```
Thread, Event
```

```
# A thread that produces data
```

```
def
```

```
producer(out_q):
```

```
    while
```

```
running:
```

```
# Produce some data
```

```
...
```

```
# Make an (data, event) pair and hand it to the consumer
```

```
evt = Event()
```

```
out_q.put((data, evt))
```

```
...
```

```
# Wait for the consumer to process the item
```

```
evt.wait()

# A thread that consumes data

def
consumer(in_q):
    while
True:

# Get some data

data, evt = in_q.get()

# Process the data

...

# Indicate completion

evt.set()
```

## 12.3.3 □□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
from queue import
```

```
Queue
```

```
from threading import
```

```
Thread
```

```
import copy
```

```
# A thread that produces data
```

```
def
```

```
producer(out_q):
```

```
    while
```

```
True:
```

```
# Produce some data
```

```

...

out_q.put(copy.deepcopy(data))

# A thread that consumes data

def
consumer(in_q):
    while
True:

# Get some data

data = in_q.get()

# Process the data

    ...

```

Queue是一个线程安全的先进先出的数据结构  
 它支持两种操作：put()和get()。  
 Queue(N)表示一个大小为N的队列。  
 put()方法用于向队列中添加元素。  
 get()方法用于从队列中移除元素。

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□

get()□put()□□□□□□□□□□□□□□□□□□□□

```
import queue

q = queue.Queue()

try
:

data = q.get(block=False)
except
queue.Empty:

...
try
:

q.put(item, block=False)
except
queue.Full:

...
try
```



```
:
data = q.get(timeout=5.0)
except
queue.Empty:
...

```

[illegible]

```
def
producer(q):
    ...
    try
:
q.put(item, block=False)
    except
queue.Full:

log.warning('queued item %r discarded!', item)
```

q.get() returns None if the queue is empty. This is a common way to check for termination flag. 12.1

```
_running = True

def
consumer(q):
    while
_running:
        try

:

item = q.get(timeout=5.0)

# Process item

...
        except
queue.Empty:
            pass
```

q.qsize() returns the number of items in the queue. q.full() and q.empty() return True if the queue is full or empty, respectively.

`q.empty()` returns `True` if the queue is empty, `False` otherwise.  
The queue is empty if there are no items in it.  
The queue is not empty if there are items in it.

## 12.4 线程局部存储

### 12.4.1 线程局部存储

线程局部存储 (Thread-Local Storage, TLS) 是一种存储模型，  
每个线程都有自己的私有存储区域，不会与其他线程共享。  
这可以避免 race condition 问题。

### 12.4.2 线程局部存储

在 Python 中，线程局部存储可以通过 `threading.local()` 实现。  
这可以确保每个线程都有自己的私有存储区域，不会与其他线程共享。  
这可以避免 race condition 问题。

```
import threading
```

```
class SharedCounter
```

```
:
```

```
    ...
```

*A counter object that can be shared by multiple threads.*

```
'''

    def
__init__(self, initial_value = 0):

self._value = initial_value

self._value_lock = threading.Lock()

    def
incr(self,delta=1):

'''

Increment the counter with locking

'''

    with
self._value_lock:

self._value += delta

    def
decr(self,delta=1):
```

```
'''
Decrement the counter with locking

'''

    with
self._value_lock:

self._value -= delta
```

with Lock — —  
 with with

## 12.4.3

# Python多线程编程

```
import threading

class SharedCounter
:
    '''
    A counter object that can be shared by multiple threads.
    '''

    def
    __init__(self, initial_value = 0):

self._value = initial_value

self._value_lock = threading.Lock()

    def
    incr(self, delta=1):

    '''
```

*Increment the counter with locking*

'''

```
self._value_lock.acquire()
```

```
self._value += delta
```

```
self._value_lock.release()
```

**def**

```
decr(self,delta=1):
```

'''

*Decrement the counter with locking*

'''

```
self._value_lock.acquire()
```

```
self._value -= delta
```

```
self._value_lock.release()
```

```

    with open('data.txt') as f:
        f.read()
    f.release()
    with open('data.txt') as f:

```

**Figure 1**

```

    threading.Lock()
RLock Semaphore RLock
"""
"""
SharedCounter

```

```
import threading
```

```
class SharedCounter
```

□ □

**///**



*A counter object that can be shared by multiple threads.*

```
'''
```

```
_lock = threading.RLock()
```

```
    def
```

```
__init__(self, initial_value = 0):
```

```
    self._value = initial_value
```

```
        def
```

```
incr(self,delta=1):
```

```
'''
```

*Increment the counter with locking*

```
'''
```

```
        with
```

```
SharedCounter._lock:
```

```
    self._value += delta
```

```
def
decr(self,delta=1):

'''

Decrement the counter with locking

'''

with
SharedCounter._lock:

self.incr(-delta)
```

decr()

counter
counter
counter

Semaphore is a mutual exclusion primitive that can be used to limit the number of threads that can access a resource at the same time. It is similar to a Lock, but it can be used to limit the number of threads that can access a resource at the same time. It is a good choice for throttling requests to a server.

Semaphore

```
from threading import
Semaphore
import urllib.request

# At most, five threads allowed to run at once

_fetch_url_sema = Semaphore(5)

def
fetch_url(url):
    with
_fetch_url_sema:
    return
urllib.request.urlopen(url)
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□

## 12.5 □□□□

### 12.5.1 □□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□

### 12.5.2 □□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
import threading

from contextlib import
contextmanager

# Thread-local state to stored information on locks already
```

```

acquired

_local = threading.local()

@contextmanager
def
acquire(*locks):

    # Sort locks by object identifier

    locks = sorted(locks, key=lambda
x: id(x))

    # Make sure lock order of previously acquired locks is not
violated

    acquired = getattr(_local, 'acquired', [])
        if
acquired and
max(id(lock) for
lock in
acquired) >= id(locks[0]):
        raise RuntimeError
('Lock Order Violation')

    # Acquire all of the locks

```

```
acquired.extend(locks)

_local.acquired = acquired
    try
:
        for
lock in
locks:

lock.acquire()
    yield

        finally
:

# Release locks in reverse order of acquisition

        for
lock in
reversed(locks):

lock.release()
    del
acquired[-len(locks):]
```



————

acquire()

acquire()

thread-local  
storage

```
import threading

x_lock = threading.Lock()
y_lock = threading.Lock()

def
    thread_1():
        while
            True:
                with
                    acquire(x_lock):
                        with
                            acquire(y_lock):
                                print
                                    ('Thread-1')
def
```





```
next(self.gen)
    File "deadlock.py", line 15, in acquire
        raise RuntimeError

("Lock Order Violation")
RuntimeError: Lock Order Violation
>>>
```

```
def acquire():
    """acquire()"""
    global ID
    ID = ID + 1
```

## 12.5.3

```
def acquire():
    """acquire()"""
    global ID
    ID = ID + 1
```

```
def acquire():
    """acquire()"""
    global ID
    ID = ID + 1
```

```
def acquire():
    """acquire()"""
    global ID
    ID = ID + 1
```



```

NSTICKS = 5
chopsticks = [threading.Lock() for
n in
range(NSTICKS)]
# Create all of the philosophers

for
n in
range(NSTICKS):

t = threading.Thread(target=philosopher,

args=(chopsticks[n], chopsticks[(n+1) % NSTICKS]))

t.start()

```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
 □□□acquire()□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

## 12.6 □□□□□□□□□□

### 12.6.1 □□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□

## 12.6.2 □□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□threading.local()□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
LazyConnection□□□□□□□□□□□□8.3□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
from socket import
socket, AF_INET, SOCK_STREAM
import threading

class LazyConnection
:
    def
__init__(self, address, family=AF_INET, type=SOCK_STREAM):

self.address = address

self.family = AF_INET
```

```

self.type = SOCK_STREAM

self.local = threading.local()

    def
__enter__(self):
    if
hasattr(self.local, 'sock'):
        raise RuntimeError

('Already connected')

self.local.sock = socket(self.family, self.type)

self.local.sock.connect(self.address)
    return

self.local.sock

    def
__exit__(self, exc_ty, exc_val, tb):

self.local.sock.close()
    del

self.local.sock

```

self.local
 threading.local()
 socket

self.local.sock

LazyConnection

```
from functools import
partial
def
test(conn):
    with
conn as
s:

s.send(b'GET /index.html HTTP/1.0\r\n
')

s.send(b'Host: www.python.org\r\n
')

s.send(b'\r\n
')

resp = b''.join(iter(partial(s.recv, 8192), b''))

    print
('Got {} bytes'.format(len(resp)))
if
```

```

__name__ == '__main__':

conn = LazyConnection(('www.python.org', 80))

t1 = threading.Thread(target=test, args=(conn,))

t2 = threading.Thread(target=test, args=(conn,))

t1.start()

t2.start()

t1.join()

t2.join()

```

```

socket.setdefaulttimeout(5)
socket.setdefaulttimeout(5)
socket.setdefaulttimeout(5)

```

## 12.6.3 网络编程

```

socket.socket(socket.AF_INET, socket.SOCK_STREAM)
socket.socket(socket.AF_INET, socket.SOCK_STREAM)
socket.socket(socket.AF_INET, socket.SOCK_STREAM)

```



socket

threading.local()  
LazyConnection

threading.local()

## 12.7

### 12.7.1

### 12.7.2

concurrent.futures  
ThreadPoolExecutor  
TCP

```

from socket import
AF_INET, SOCK_STREAM, socket
from concurrent.futures import
ThreadPoolExecutor
def
echo_client(sock, client_addr):
    '''
    Handle a client connection
    '''
    print
('Got connection from', client_addr)
    while
True:
msg = sock.recv(65536)
    if not
msg:
        break

sock.sendall(msg)
print

```

```
('Client closed connection')

sock.close()

def
echo_server(addr):

pool = ThreadPoolExecutor(128)

sock = socket(AF_INET, SOCK_STREAM)

sock.bind(addr)

sock.listen(5)
    while
True:

client_sock, client_addr = sock.accept()

pool.submit(echo_client, client_sock, client_addr)
echo_server(('',15000))
```

Queue

```
from socket import
socket, AF_INET, SOCK_STREAM
```

```
from threading import
Thread
from queue import
Queue
def
echo_client(q):
    '''
    Handle a client connection
    '''

    sock, client_addr = q.get()
    print
    ('Got connection from', client_addr)
    while
    True:
        msg = sock.recv(65536)
        if not
        msg:
            break
```

```
sock.sendall(msg)
    print
('Client closed connection')

sock.close()

    def
echo_server(addr, nworkers):

# Launch the client workers

q = Queue()
    for
n in
range(nworkers):

t = Thread(target=echo_client, args=(q,))

t.daemon = True

t.start()

# Run the server

sock = socket(AF_INET, SOCK_STREAM)

sock.bind(addr)
```

```
sock.listen(5)
    while
True:

client_sock, client_addr = sock.accept()

q.put((client_sock, client_addr))

echo_server('',15000), 128)
```

ThreadPoolExecutor

```
from concurrent.futures import
```

```
ThreadPoolExecutor
import urllib.request
```

```
def
```

```
fetch_url(url):
```

```
u = urllib.request.urlopen(url)
```

```
data = u.read()
    return
```

```
data
```

```

pool = ThreadPoolExecutor(10)
# Submit work to the pool

a = pool.submit(fetch_url, 'http://www.python.org')
b = pool.submit(fetch_url, 'http://www.pypy.org')

# Get the results back

x = a.result()
y = b.result()

```

a b  
 a.result()

## 12.7.3

```

from threading import
Thread
from socket import
socket, AF_INET, SOCK_STREAM

def
echo_client(sock, client_addr):

```

```
'''
```

*Handle a client connection*

```
'''
```

```
    print
```

```
('Got connection from', client_addr)
```

```
    while
```

```
True:
```

```
msg = sock.recv(65536)
```

```
    if not
```

```
msg:
```

```
        break
```

```
sock.sendall(msg)
```

```
    print
```

```
('Client closed connection')
```

```
sock.close()
```

```
def
```

```
echo_server(addr, nworkers):
```

```
# Run the server
```





Python 解释器在启动时会为每个线程分配一定的堆内存，  
并会设置一个全局解释器锁（GIL）来保证 CPU 的利用率 [2]  
——Python 解释器通过 GIL 来保证 CPU 的利用率  
并会设置 I/O 的阻塞操作

Python 解释器在启动时会为每个线程分配一定的堆内存，  
OS X 系统下 Python 解释器默认分配的堆内存为 9  
GB，而 Linux 系统下 Python 解释器默认分配的堆内存为  
8 MB，而 Windows 系统下 Python 解释器默认分配的堆内存为  
70 MB，而 9 GB 的堆内存对于大多数系统来说都是不够的，  
因此可以通过 `threading.stack_size()` 来设置线程的堆内存大小

```
import threading

threading.stack_size(65536)
```

Python 解释器默认分配的堆内存为 210 MB，而 210 MB 的堆内存对于大多数系统来说都是不够的，  
因此可以通过 `threading.stack_size()` 来设置线程的堆内存大小，  
而 40968192 的堆内存对于大多数系统来说都是不够的

## 12.8 多线程编程

## 12.8.1 并行

并行处理可以充分利用CPU资源，提高程序运行效率。CPU资源利用率高，程序运行速度快。

## 12.8.2 多线程

concurrent.futures模块提供了ProcessPoolExecutor和ThreadPoolExecutor两个类，分别用于创建进程池和线程池。Python程序可以通过这两个类实现多线程或并行处理，提高程序效率。

gzip模块提供了gzip.open()函数，用于打开gzip压缩文件。Apache Web服务器使用gzip模块对静态文件进行压缩，提高传输效率。

```
logs/

20120701.log.gz

20120702.log.gz

20120703.log.gz

20120704.log.gz

20120705.log.gz

20120706.log.gz
```

...

robots.txt

```
124.115.6.12 - - [10/Jul/2012:00:18:50 -0500] "GET /robots.txt
..." 200 71
210.212.209.67 - - [10/Jul/2012:00:18:51 -0500] "GET /ply/
..." 200 11875
210.212.209.67 - - [10/Jul/2012:00:18:51 -0500] "GET
/favicon.ico ..." 404 369
61.135.216.105 - - [10/Jul/2012:00:20:04 -0500] "GET
/blog/atom.xml ..." 304 -
...
```

robots.txt

```
# findrobots.py
```

```
import gzip
```

```
import io
```

```
import glob
```

```
def
```

```
find_robots(filename):
```

```

'''

Find all of the hosts that access robots.txt in a single log
file

'''

robots = set()
    with
gzip.open(filename) as
f:
    for
line in
io.TextIOWrapper(f,encoding='ascii'):

fields = line.split()
    if
fields[6] == '/robots.txt':

robots.add(fields[0])
    return
robots
def
find_all_robots(logdir):

```

```
'''
```

*Find all hosts across and entire sequence of files*

```
'''
```

```
files = glob.glob(logdir+'/*.log.gz')
```

```
all_robots = set()
```

```
    for
```

```
robots in
```

```
map(find_robots, files):
```

```
all_robots.update(robots)
```

```
    return
```

```
all_robots
```

```
if
```

```
__name__ == '__main__':
```

```
robots = find_all_robots('logs')
```

```
    for
```

```
ipaddr in
```

```
robots:
```

```
    print
```

```
(ipaddr)
```

map-reduce [3] 实现  
find\_robots() 函数遍历所有文件并返回所有发现的机器人  
find\_all\_robots() 函数返回所有发现的机器人  
all\_robots

使用 CPU 并行处理  
—— 使用 map() 函数并行处理  
concurrent.futures 模块  
实现

```
# findrobots.py

import gzip

import io

import glob

from concurrent import
futures

def
find_robots(filename):

    ...
```

*Find all of the hosts that access robots.txt in a single log file*

```
'''

robots = set()
    with
gzip.open(filename) as
f:
    for
line in
io.TextIOWrapper(f,encoding='ascii'):

fields = line.split()
    if
fields[6] == '/robots.txt':

robots.add(fields[0])
    return
robots

def
find_all_robots(logdir):

'''
```



*Find all hosts across and entire sequence of files*

```
'''

files = glob.glob(logdir+'/*.log.gz')

all_robots = set()
    with
futures.ProcessPoolExecutor() as
pool:
    for
robots in
pool.map(find_robots, files):

all_robots.update(robots)
    return
all_robots
if
__name__ == '__main__':

robots = find_all_robots('logs')
    for
ipaddr in
```



ProcessPoolExecutor(N) Python N CPU Python ProcessPoolExecutor(N) with

map() pool.map()

```
# A function that performs a lot of work
```

```
def
```

```
work(x):
```

```
...
```

```
    return
```

```
result
```

```
# Nonparallel code
```

```
results = map(work, data)
```

```
# Parallel implementation
```

```
with
```

```
ProcessPoolExecutor() as
```

```
pool:

results = pool.map(work, data)
```

□□□□□□□□□□□□pool.submit()□□□□□□□□□□  
□□□□□□

```
# Some function

def
work(x):

...
    return
result

with
ProcessPoolExecutor() as
pool:

...

# Example of submitting work to the pool

future_result = pool.submit(work, arg)
```

```
# Obtaining the result (blocks until done)
```

```
r = future_result.result()  
...
```

```
Future  
result()  

```

```
def  
when_done(r):  
    print  
    ('Got:', r.result())  
with  
    ProcessPoolExecutor() as  
    pool:  
  
    future_result = pool.submit(work, arg)  
  
    future_result.add_done_callback(when_done)
```

Future 객체를 반환하며  
result()로 결과를 가져옴

Parallelism은 Task를 분할하여  
동시에 실행하는 것

- Task를 분할하여  
실행
- Task를 분할하여  
실행
- Task를 분할하여  
pickle로 직렬화하여  
실행
- Task를 분할하여  
pure-function로 실행
- Task를 분할하여  
UNIX fork()로 실행  
Python fork()로 실행  
Windows로 실행  
pool.map()로  
pool.submit()로 실행

- Python 2.7.10 版本中，GIL 的实现方式与之前版本有所不同，导致在某些情况下，多线程程序的性能可能会受到影响。

## 12.9 Python 的 GIL

### 12.9.1 简介

Python 的 GIL 是全局解释器锁（Global Interpreter Lock）的缩写，它的作用是保证在任意时刻，只有一个线程在解释器中执行 Python 代码。

### 12.9.2 多线程

Python 的多线程实现依赖于 GIL。在 CPython 解释器中，GIL 的作用是保证在任意时刻，只有一个线程在解释器中执行 Python 代码。Python 的多线程实现依赖于 GIL。在 CPython 解释器中，GIL 的作用是保证在任意时刻，只有一个线程在解释器中执行 Python 代码。Python 的多线程实现依赖于 GIL。在 CPython 解释器中，GIL 的作用是保证在任意时刻，只有一个线程在解释器中执行 Python 代码。

Python 的多线程实现依赖于 GIL。在 CPython 解释器中，GIL 的作用是保证在任意时刻，只有一个线程在解释器中执行 Python 代码。Python 的多线程实现依赖于 GIL。在 CPython 解释器中，GIL 的作用是保证在任意时刻，只有一个线程在解释器中执行 Python 代码。Python 的多线程实现依赖于 GIL。在 CPython 解释器中，GIL 的作用是保证在任意时刻，只有一个线程在解释器中执行 Python 代码。

Pythonの性能を向上させるには、  
いくつかの方法があります。

CPUの性能を向上させるには、  
Pythonの性能を向上させるには、  
C言語の性能を向上させるには、  
NumPyの性能を向上させるには、  
JITコンパイラPyPyの性能を向上させるには、  
Python 3の性能を向上させるには、

CPUの性能を向上させるには、  
GILの性能を向上させるには、  
GILの性能を向上させるには、  
CPUの性能を向上させるには、  
C言語の性能を向上させるには、  
C言語の性能を向上させるには、  
[4]

GILの性能を向上させるには、  
Pythonの性能を向上させるには、  
multiprocessingの性能を向上させるには、

```
# Performs a large calculation (CPU bound)
```

```
def
```



```

some_work(args):
    ...
    return

result

# A thread that calls the above function

def

some_thread():
    while

True:
    ...
    r = some_work(args)
    ...

```

□□□□□□□□□□□□□□□□□□□□□□□□

```

# Processing pool (see below for initiazation)

pool = None

# Performs a large calculation (CPU bound)

def

some_work(args):
    ...
    return

result

# A thread that calls the above function

```

```

def
some_thread():
    while
True:
    ...
    r = pool.apply(some_work, (args))
    ...

# Initiaze the pool

if
__name__ == '__main__':
    import multiprocessing

    pool = multiprocessing.Pool()

```

Python 2.7 版本中，GIL 是 Python 解释器的一部分，它负责管理 CPU 的线程。GIL 的存在使得 Python 无法充分利用多核 CPU 的性能。

在 Python 3 中，GIL 仍然存在，但已经不再强制。Python 3 的 C 扩展模块可以绕过 GIL，从而实现多线程并行。



pickle  
pickle

singleton  
singleton

C Python  
Python C  
C Python Python  
Python C API C  
C

GIL  
C  
PyPy

C GIL 15.7  
15.10

## 12.10 Actor

## 12.10.1 ☐

```
actor actor

```

## 12.10.2 ☐☐☐☐

[illegible]

actor

```
from queue import
Queue
from threading import
Thread, Event

# Sentinel used for shutdown

class ActorExit
(Exception
```

```

):
    pass

class Actor
:
    def
__init__(self):
    self._mailbox = Queue()

    def
send(self, msg):
    '''

        Send a message to the actor

    '''

    self._mailbox.put(msg)

    def
recv(self):
    '''

        Receive an incoming message

    '''

    msg = self._mailbox.get()
    if
msg is
ActorExit:

```

```

            raise
    ActorExit()
    return

msg

    def
close(self):
    '''

        Close the actor, thus shutting it down

        ...

        self.send(ActorExit)

    def
start(self):
    '''

        Start concurrent execution

        ...

        self._terminated = Event()
        t = Thread(target=self._bootstrap)
        t.daemon = True
        t.start()

    def
_bootstrap(self):
    try

:
        self.run()

```

```

        except
ActorExit:
            pass

        finally

:
            self._terminated.set()

    def
join(self):
    self._terminated.wait()

    def
run(self):
    '''

        Run method to be implemented by the user

        '''

        while
True:
            msg = self.recv()

# Sample ActorTask

class PrintActor
(Actor):
    def
run(self):
        while
True:

```



```

        msg = self.recv()
        print

('Got:', msg)

# Sample use

p = PrintActor()
p.start()
p.send('Hello')
p.send('World')
p.close()
p.join()

```

actor send() actor  
 actor send() actor  
 actor close() actor  
 ActorExit actor Actor  
 actor run() actor  
 ActorExit actor  
 ActorExit recv() actor

actor

```

def
print_actor():
    while
True:
        try

```

```

:
    msg = yield
# Get a message

    print
('Got:', msg)
    except GeneratorExit
:
    print
('Actor terminating')
# Sample use

p = print_actor()
next(p)          # Advance to the yield (ready to receive)

p.send('Hello')
p.send('World')
p.close()

```

## 12.10.3 消息

actor 消息接收器接收消息并处理它们。消息接收器可以发送消息给其他 actor。消息接收器可以接收来自其他 actor 的消息。消息接收器可以接收来自其他 actor 的消息。消息接收器可以接收来自其他 actor 的消息。

```

class TaggedActor

```

```

(Actor):
    def
run(self):
    while
True:
    tag, *payload = self.recv()
    getattr(self, 'do_'+tag)(*payload)

    # Methods corresponding to different message tags

    def
do_A(self, x):
    print
('Running A', x)

    def
do_B(self, x, y):
    print
('Running B', x, y)

# Example

a = TaggedActor()
a.start()
a.send(('A', 1))          # Invokes do_A(1)

a.send(('B', 2, 3))       # Invokes do_B(2,3)

```

actor
Result

```
from threading import
```

```
Event
```

```
class Result
```

```
:
```

```
    def
```

```
    __init__(self):
```

```
        self._evt = Event()
```

```
        self._result = None
```

```
    def
```

```
    set_result(self, value):
```

```
        self._result = value
```

```
        self._evt.set()
```

```
    def
```

```
    result(self):
```

```
        self._evt.wait()
```

```
        return
```

```
    self._result
```

```
class Worker
```

```
    (Actor):
```

```
        def
```

```
        submit(self, func, *args, **kwargs):
```

```
            r = Result()
```

```
            self.send((func, args, kwargs, r))
```

```
            return
```

```
        r
```

```
        def
```

```
        run(self):
```

```
            while
```

```
            True:
```



□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
from collections import
defaultdict

class Exchange

:
    def

__init__(self):
    self._subscribers = set()

    def

attach(self, task):
    self._subscribers.add(task)

    def

detach(self, task):
    self._subscribers.remove(task)

    def

send(self, msg):
    for

subscriber in

self._subscribers:
    subscriber.send(msg)

# Dictionary of all created exchanges

_exchanges = defaultdict(Exchange)

# Return the Exchange instance associated with a given name
```

```
def
```

```
get_exchange(name):  
    return
```

```
_exchanges[name]
```

```
    """  
    Return the exchange object for the given name.  
    """  
    get_exchange()"""  
    Exchange"""
```

```
    """
```

```
# Example of a task. Any object with a send() method
```

```
class Task
```

```
:
```

```
    ...  
    def
```

```
send(self, msg):  
    ...
```

```
task_a = Task()  
task_b = Task()
```

```
# Example of getting an exchange
```

```
exc = get_exchange('name')
```

```
exc.attach(task_a)
exc.attach(task_b)
```

```
exc.send( 'msg1' )
exc.send( 'msg2' )
```

```
exc.detach(task_a)
exc.detach(task_b)
```

### 12.11.3 ☐ ☐

logging



1. 在消息中，我们使用 `fan-out` 来指定消息的接收者。

```

class DisplayMessages
:
    def
__init__(self):
    self.count = 0
    def
send(self, msg):
    self.count += 1
    print
('msg[{}]: {!r}'.format(self.count, msg))

exc = get_exchange('name')
d = DisplayMessages()
exc.attach(d)

```

2. 在消息中，我们使用 `actor` 来指定消息的接收者。

3. 在消息中，我们使用 `send()` 来指定消息的接收者。

```

exc = get_exchange('name')
exc.attach(some_task)
try
:
...
finally
:
    exc.detach(some_task)

```

1. 在任务完成前，调用 detach() 方法，将任务从交换器中移除。

2. 在任务完成后，调用 subscribe() 方法，将任务添加到交换器中。

```

from contextlib import
contextmanager
from collections import
defaultdict

class Exchange
:
    def
__init__(self):
        self._subscribers = set()

    def
attach(self, task):
        self._subscribers.add(task)

    def

```

```
detach(self, task):
    self._subscribers.remove(task)
```

```
@contextmanager
```

```
def
```

```
subscribe(self, *tasks):
    for
```

```
task in
```

```
tasks:
    self.attach(task)
    try
```

```
:
```

```
    yield
```

```
    finally
```

```
:
```

```
    for
```

```
task in
```

```
tasks:
    self.detach(task)
```

```
    def
```

```
send(self, msg):
    for
```

```
subscriber in
```

```
self._subscribers:
    subscriber.send(msg)
```

```
# Dictionary of all created exchanges
```

```
_exchanges = defaultdict(Exchange)
```

```

# Return the Exchange instance associated with a given name

def
get_exchange(name):
    return
    _exchanges[name]

# Example of using the subscribe() method

exc = get_exchange('name')
with
exc.subscribe(task_a, task_b):
    ...
    exc.send('msg1')
    exc.send('msg2')
    ...

# task_a and task_b detached here

```

```

    □□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```

## 12.12 □□□□□□□□□□□□□□□□

### 12.12.1 □□

yield语句是Python中一个特殊的语句，它允许你编写一个函数，该函数可以返回一个序列的迭代器对象。这个对象可以在每次迭代时返回下一个元素，而不需要一次性返回整个序列。这通常用于处理大型数据集或生成器函数。

## 12.12.2 生成器函数

生成器函数是一种特殊的函数，它使用yield语句来返回一个序列的迭代器对象。与普通的函数不同，生成器函数在每次调用时只会返回一个值，而不是整个序列。这使得生成器函数非常适合用于处理大型数据集或需要惰性求值的场景。生成器函数通常用于生成器表达式、range函数、zip函数等。

生成器函数可以看作是一个特殊的函数，它返回一个生成器对象。这个对象可以在每次迭代时返回下一个元素，而不需要一次性返回整个序列。这通常用于处理大型数据集或生成器函数。

```
# Two simple generator functions
```

```
def
```

```
countdown(n):
```

```
    while
```

```
n > 0:
```

```
    print
```

```
    ('T-minus', n)
```

```
    yield
```

```
    n -= 1
```

```
    print
```

```
    ('Blastoff!')
```

```
def
```

```
countup(n):
```

```

    x = 0
    while
x < n:
    print
('Counting up', x)
    yield

    x += 1

```

yield

```

from collections import
deque
class TaskScheduler
:
    def
__init__(self):
    self._task_queue = deque()

    def
new_task(self, task):
    '''
        Admit a newly started task to the scheduler
    '''

```

```

        self._task_queue.append(task)

    def
run(self):
    '''

        Run until there are no more tasks

        ...

    while
self._task_queue:
    task = self._task_queue.popleft()
    try
:
        # Run until the next yield statement

        next(task)
        self._task_queue.append(task)
    except StopIteration
:
        # Generator is no longer executing

        pass

# Example use

sched = TaskScheduler()
sched.new_task(countdown(10))
sched.new_task(countdown(5))
sched.new_task(countup(15))
sched.run()

```

TaskScheduler  
yield  
—  
yield

```
T-minus 10
T-minus 5
Counting up 0
T-minus 9
T-minus 4
Counting up 1
T-minus 8
T-minus 3
Counting up 2
T-minus 7
T-minus 2
...
```

yield  
yield  
yield

actor

actor

```
from collections import
deque
```



```

class ActorScheduler
:
    def
__init__(self):
    self._actors = { }          # Mapping of names to
actors

    self._msg_queue = deque()    # Message queue

    def
new_actor(self, name, actor):
    '''

        Admit a newly started actor to the scheduler and give
it a name

        '''

    self._msg_queue.append((actor, None))
    self._actors[name] = actor

    def
send(self, name, msg):
    '''

        Send a message to a named actor

        '''

    actor = self._actors.get(name)
    if

```

```

actor:
    self._msg_queue.append((actor,msg))

    def
run(self):
    '''

    Run as long as there are pending messages.

    '''

    while
self._msg_queue:
    actor, msg = self._msg_queue.popleft()
    try
:
        actor.send(msg)
    except StopIteration
:
        pass

# Example use

if
__name__ == '__main__':
    def
printer():
    while
True:
        msg = yield

```

```

        print
('Got:', msg)

    def
counter(sched):
    while
True:
        # Receive the current count

        n = yield

        if
n == 0:
            break

        # Send to the printer task

        sched.send('printer', n)
        # Send the next count to the counter task
(recursive)

        sched.send('counter', n-1)

sched = ActorScheduler()
# Create the initial actors

sched.new_actor('printer', printer())
sched.new_actor('counter', counter(sched))

# Send an initial message to the counter to initiate

sched.send('counter', 10000)
sched.run()

```

□□□□□□□□□□□□□□□□□□□□□□□□□□  
 □□□□□□□□□□□□□□□□□□□□□□□□□□□□  
 □□□□□**counter**□□□□□□□□□□□□□□□□□□□□  
 □□□□**Python**□□□□□□□□□□□□□□□□□□□□

A diagram consisting of two rows of squares. The top row contains 20 squares, and the bottom row contains 3 squares.

```
from collections import
deque
from select import
select

# This class represents a generic yield event in the scheduler

class YieldEvent
:
    def
handle_yield(self, sched, task):
    pass

    def
handle_resume(self, sched, task):
    pass
```

```

# Task Scheduler

class Scheduler
:
    def
__init__(self):
    self._numtasks = 0          # Total num of tasks

    self._ready = deque()      # Tasks ready to run

    self._read_waiting = {}    # Tasks waiting to
read

    self._write_waiting = {}   # Tasks waiting to write

# Poll for I/O events and restart waiting tasks

def
_iopoll(self):
    rset,wset,eset = select(self._read_waiting,
                             self._write_waiting,[])
    for
r in
rset:
    evt, task = self._read_waiting.pop(r)
    evt.handle_resume(self, task)
    for
w in
wset:
    evt, task = self._write_waiting.pop(w)
    evt.handle_resume(self, task)

```

**def**

new(self, task):  
 '''

*Add a newly started task to the scheduler*

'''

self.\_ready.append((task, None))  
self.\_numtasks += 1

**def**

add\_ready(self, task, msg=None):  
 '''

*Append an already started task to the ready queue.*

*msg is what to send into the task when it resumes.*

'''

self.\_ready.append((task, msg))

*# Add a task to the reading set*

**def**

\_read\_wait(self, fileno, evt, task):  
 self.\_read\_waiting[fileno] = (evt, task)

*# Add a task to the write set*

**def**

```

_write_wait(self, fileno, evt, task):
    self._write_waiting[fileno] = (evt, task)

def
run(self):
    '''

    Run the task scheduler until there are no tasks

    '''

    while
self._numtasks:
    if not
self._ready:
        self._iopoll()
        task, msg = self._ready.popleft()
        try

:
            # Run the coroutine to the next yield

            r = task.send(msg)
            if

isinstance(r, YieldEvent):
                r.handle_yield(self, task)
            else

:
                raise RuntimeError

('unrecognized yield event')
            except StopIteration

:
                self._numtasks -= 1

```

*# Example implementation of coroutine-based socket I/O*

### **class ReadSocket**

(YieldEvent):

**def**

**\_\_init\_\_**(self, sock, nbytes):

    self.sock = sock

    self.nbytes = nbytes

**def**

handle\_yield(self, sched, task):

    sched.\_read\_wait(self.sock.fileno(), self, task)

**def**

handle\_resume(self, sched, task):

    data = self.sock.recv(self.nbytes)

    sched.add\_ready(task, data)

### **class WriteSocket**

(YieldEvent):

**def**

**\_\_init\_\_**(self, sock, data):

    self.sock = sock

    self.data = data

**def**

handle\_yield(self, sched, task):

    sched.\_write\_wait(self.sock.fileno(), self, task)

**def**

handle\_resume(self, sched, task):

    nsent = self.sock.send(self.data)

    sched.add\_ready(task, nsent)

### **class AcceptSocket**

(YieldEvent):

**def**



```

__init__(self, sock):
    self.sock = sock
    def

handle_yield(self, sched, task):
    sched._read_wait(self.sock.fileno(), self, task)
    def

handle_resume(self, sched, task):
    r = self.sock.accept()
    sched.add_ready(task, r)

# Wrapper around a socket object for use with yield

class Socket
(object):
    def

__init__(self, sock):
    self._sock = sock
    def

recv(self, maxbytes):
    return

ReadSocket(self._sock, maxbytes)
def

send(self, data):
    return

WriteSocket(self._sock, data)
def

accept(self):
    return

AcceptSocket(self._sock)
def

__getattr__(self, name):
    return

```

```

getattr(self._sock, name)

if

__name__ == '__main__':
    from socket import

socket, AF_INET, SOCK_STREAM
    import time


    # Example of a function involving generators. This should

    # be called using line = yield from readline(sock)


    def

readline(sock):
    chars = []
    while

True:
        c = yield

sock.recv(1)
        if not

c:
            break

            chars.append(c)
            if

c == b'\n

':
            break


    return

```

```

b''.join(chars)

# Echo server using generators

class EchoServer
:
    def

__init__(self,addr,sched):
    self.sched = sched
    sched.new(self.server_loop(addr))

    def

server_loop(self,addr):
    s = Socket(socket(AF_INET,SOCK_STREAM))
    s.bind(addr)
    s.listen(5)
    while

True:
        c,a = yield

s.accept()
    print

('Got connection from ', a)
        self.sched.new(self.client_handler(Socket(c)))

    def

client_handler(self,client):
    while

True:
        line = yield from readline

(client

)

        if not

line:

```

```

        break

    line = b'GOT:' + line
    while
line:
        nsent = yield
client.send(line)
        line = line[nsent:]
    client.close()
    print
('Client closed')

sched = Scheduler()
EchoServer(('',16000),sched)
sched.run()

```

deque
 I/O
 I/O

## 12.12.3

yield

```

def
some_generator():
    ...

```



close() GeneratorExit  
yield GeneratorExit  
throw() yield  
GeneratorExit

yield from  
subroutine procedure  
yield from  
yield from yield from PEP  
380 <http://www.python.org/dev/peps/pep-0380>

CPU I/O  
Python  
PEP 342  
<http://www.python.org/dev/peps/pep-0342> David Beazley PyCon 2009  
“A Curious Course On Coroutines and

Concurrency” <http://www.dabeaz.com/coroutines> □□

# PEP 3156

<http://www.python.org/dev/peps/pep-3156>

Python

<http://www.gevent.org>

<http://pypi.python.org/pypi/gevent>

Stackless Python

<http://www.stackless.com>

## 12.13

### 12.13.1 □□

## 12.13.2 ☐☐☐☐

[illegible]

socket socket  
select()  
[]

```
import queue

import socket

import os

class PollableQueue
(queue.Queue):
    def
__init__(self):
    super().__init__()
    # Create a pair of connected sockets

    if
os.name == 'posix':
        self._putsocket, self._getsocket =
socket.socketpair()
    else
:
        # Compatibility on non-POSIX systems

        server = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        server.bind(('127.0.0.1', 0))
        server.listen(1)
        self._putsocket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
```



```
self._putsocket.connect(server.getsockname())
self._getsocket, _ = server.accept()
server.close()
```

**def**

```
fileno(self):
    return
```

```
self._getsocket.fileno()
```

**def**

```
put(self, item):
    super().put(item)
    self._putsocket.send(b'x')
```

**def**

```
get(self):
    self._getsocket.recv(1)
    return
```

```
super().get()
```

Queue socket UNIX socketpair() socket Windows socket socket socket socket socket socket socket get() put() socket I/O put() socket get() socket

`fileno()` returns an integer that can be passed to `select()` to monitor for activity. `fileno()` returns the file descriptor of the socket.

Example:

```
import select

import threading

def
consumer(queues):
    '''
    Consumer that reads data on multiple queues simultaneously
    '''

    while
True:
        can_read, _, _ = select.select(queues, [], [])
        for
r in
can_read:
            item = r.get()
            print
('Got:', item)
```





```

event_loop(sockets, queues):
    while
True:
    # polling with a timeout

    can_read, _, _ = select.select(sockets, [], [], 0.01)
    for

r in
can_read:
    handle_read(r)
    for

q in
queues:
    if not

q.empty():
    item = q.get()
    print

('Got:', item)

```

socket select()
 I/O socket

## 12.14 UNIX

### 12.14.1

UNIX

### 12.14.2

```
#!/usr/bin/env python3
```

```
# daemon.py
```

```
import os
```

```
import sys
```

```
import atexit
```

```
import signal
```

```

def
daemonize(pidfile, *, stdin='/dev/null',
           stdout='/dev/null',
           stderr='/dev/null'):

    if
os.path.exists(pidfile):
        raise RuntimeError
('Already running')

    # First fork (detaches from parent)

    try
:
        if
os.fork() > 0:
        raise SystemExit
(0) # Parent exit

    except OSError as
e:
        raise RuntimeError
('fork #1 failed.')

    os.chdir('/')
    os.umask(0)
    os.setsid()
    # Second fork (relinquish session leadership)

    try
:
        if

```

```
os.fork() > 0:
    raise SystemExit

(0)
    except OSError as
e:
    raise RuntimeError

('fork #2 failed.')

    # Flush I/O buffers

    sys.stdout.flush()
    sys.stderr.flush()

    # Replace file descriptors for stdin, stdout, and stderr

    with
open(stdin, 'rb', 0) as
f:
    os.dup2(f.fileno(), sys.stdin.fileno())
    with
open(stdout, 'ab', 0) as
f:
    os.dup2(f.fileno(), sys.stdout.fileno())
    with
open(stderr, 'ab', 0) as
f:
    os.dup2(f.fileno(), sys.stderr.fileno())

    # Write the PID file

    with
open(pidfile, 'w') as
```



```

f:
    print
(os.getpid(),file=f)
    # Arrange to have the PID file removed on exit/signal

    atexit.register(lambda
: os.remove(pidfile))
    # Signal handler for termination (required)

    def
sigterm_handler(signo, frame):
    raise SystemExit

(1)
    signal.signal(signal.SIGTERM, sigterm_handler)
def
main():
    import time

    sys.stdout.write('Daemon started with pid {}\n'
'.format(os.getpid()))
    while
True:
        sys.stdout.write('Daemon Alive! {}\n'
'.format(time.ctime()))
        time.sleep(10)
if
__name__ == '__main__':

```

```

    PIDFILE = '/tmp/daemon.pid'

    if
len(sys.argv) != 2:
    print
('Usage: {} [start|stop]'.format(sys.argv[0]),
file=sys.stderr)
    raise SystemExit

(1)

    if
sys.argv[1] == 'start':
    try

:
        daemonize(PIDFILE,
                    stdout='/tmp/daemon.log',
                    stderr='/tmp/dameon.log')
    except RuntimeError as

e:
    print

(e, file=sys.stderr)
    raise SystemExit

(1)

    main()

elif
sys.argv[1] == 'stop':
    if
os.path.exists(PIDFILE):
        with
open(PIDFILE) as
f:

```



```
bash % daemon.py stop
bash %
```

## 12.14.3 子进程

通常我们使用`daemonize()`函数来创建子进程并设置环境变量。该函数接受以下参数：`daemonize()`函数接受以下参数：`keyword-only`参数。该函数接受以下参数：

```
daemonize('daemon.pid',
          stdin='/dev/null',
          stdout='/tmp/daemon.log',
          stderr='/tmp/daemon.log')
```

该函数接受以下参数：

*# Illegal. Must use keyword arguments*

```
daemonize('daemon.pid',
          '/dev/null', '/tmp/daemon.log', '/tmp/daemon.log')
```

该函数接受以下参数：`os.fork()`函数。该函数接受以下参数：

os.setsid() 可以设置进程成为 session leader，  
从而避免僵尸进程的产生。调用 os.setsid() 后，  
进程会成为一个新的 session 的 leader，  
其 session ID 等于进程 ID。这通常用于后台运行  
的守护进程。

os.chdir() 和 os.umask(0) 分别用于改变当前  
工作目录和设置文件创建时的默认权限。  
os.chdir() 用于切换到指定的目录，  
os.umask(0) 用于设置文件创建时的默认权限为 0。

os.fork() 用于创建子进程。调用 os.fork() 后，  
会创建一个新的子进程，该子进程是父进程的副本。  
父进程和子进程可以分别执行不同的任务，  
从而实现并行处理。

在 Python 中，sys.stdout 和 sys.\_\_stdout\_\_ 都是指向  
标准输出流的对象。sys.stdout 是 Python 解释器  
使用的标准输出流，而 sys.\_\_stdout\_\_ 是操作系统  
提供的标准输出流。通过 os.dup2() 函数，  
可以将 sys.stdout 与 sys.\_\_stdout\_\_ 关联起来，  
从而确保 Python 程序能够正确地向操作系统  
的标准输出流写入数据。这通常用于实现  
与操作系统的 I/O 交互。

process.pid 是进程的 ID 号。调用 `daemonize()` 函数，使进程成为守护进程。  
调用 `atexit.register()` 函数，注册 Python 进程退出时调用的函数。  
调用 `SIGTERM` 信号，使进程退出。调用 `SystemExit()` 函数，使进程退出。  
调用 `atexit.register()` 函数，注册 Python 进程退出时调用的函数。  
调用 `stop` 函数，使进程退出。

W. Richard Stevens  
Stephen A. Rago *Advanced  
Programming in the UNIX Environment*  
2nd Edition Addison-Wesley 2005  
C 语言与 POSIX 系统编程 Python  
POSIX 系统编程 Python

---

[1] 进程的 ID 号。调用 `process.pid` 函数，使进程退出。

[2] 进程的 ID 号。调用 `process.pid` 函数，使进程退出。

[3] 进程的 ID 号。调用 `map` 函数，使进程退出。

[illegible]

[5] socket  
——

**13**

Python shell

# 13.1

### 13.1.1 ☐ ☐

## 13.1.2 ☐☐☐☐

# Python fileinput

```
#!/usr/bin/env python3
```

```
import fileinput
```



```
with
fileinput.input() as
    f_input:
        for
            line in
                f_input:
                    print
                        (line, end='')
```

[illegible]

```
$ ls | ./filein.py # Prints a directory listing to
stdout.

$ ./filein.py /etc/passwd # Reads /etc/passwd to
stdout.

$ ./filein.py < /etc/passwd # Reads /etc/passwd to stdout.
```

### 13.1.3 □□

fileinput.input()FileInput  
FileInput  
FileInput  
FileInput

```
>>> import fileinput

>>> with
fileinput.input('/etc/passwd') as
f:
>>>     for
line in
f:
...     print
(f.filename(), f.lineno(), line, end='')
...
/etc/passwd 1 ##

/etc/passwd 2 # User Database

/etc/passwd 3 #

<other output omitted>
```

FileInput

## 13.2 异常退出

### 13.2.1 异常退出

在 Python 中，异常退出可以通过调用 `sys.exit()` 来实现。这会导致程序立即终止并返回一个退出状态码。

### 13.2.2 异常退出

在 Python 中，异常退出可以通过调用 `SystemExit` 来实现。这会导致程序立即终止并返回一个退出状态码。

```
raise SystemExit('It failed!')
```

在 Python 中，异常退出可以通过调用 `sys.stderr` 来实现。这会导致程序立即终止并返回一个退出状态码。

### 13.2.3 异常退出

在 Python 中，异常退出可以通过调用 `sys.stderr` 来实现。这会导致程序立即终止并返回一个退出状态码。

```
import sys

sys.stderr.write('It failed!\n')
```

```
'')
raise SystemExit

(1)
```

```
import sys.stderr
sys.stderr.write('SystemExit()')
sys.stderr.flush()
```

## 13.3 模块

### 13.3.1 模块

```
import sys
sys.argv
```

### 13.3.2 模块

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument('...')
```

```
# search.py
```

```
'''
```

```
Hypothetical command-line tool for searching a collection of
```

*files for one or more text patterns.*

*'''*

**import** argparse

parser = argparse.ArgumentParser(description='Search some files')

parser.add\_argument(dest='filenames', metavar='filename', nargs='\*')

parser.add\_argument('-p', '--pat', metavar='pattern', required=True,  
dest='patterns', action='append',  
help='text pattern to search for')

parser.add\_argument('-v', dest='verbose', action='store\_true',  
help='verbose mode')

parser.add\_argument('-o', dest='outfile', action='store',  
help='output file')

parser.add\_argument('--speed', dest='speed', action='store',  
choices={'slow', 'fast'},  
default='slow',  
help='search speed')

args = parser.parse\_args()

*# Output the collected arguments*

**print**

(args.filenames)

**print**

(args.patterns)

**print**

```
(args.verbose)
print

(args.outfile)
print

(args.speed)
```

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

```
bash % python3 search.py -h
usage: search.py [-h] [-p pattern] [-v] [-o OUTFILE] [--speed
{slow,fast}]
                [filename [filename ...]]

Search some files

positional arguments:
  filename

optional arguments:
  -h, --help            show this help message and exit
  -p pattern, --pat pattern
                        text pattern to search for
  -v                    verbose mode
  -o                    OUTFILE output file
  --speed {slow,fast}  search speed
```

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

```
print()XXXXXX
```

```
bash % python3 search.py foo.txt bar.txt
usage: search.py [-h] -p pattern [-v] [-o OUTFILE] [--speed
{fast,slow}]
                [filename [filename ...]]
search.py: error: the following arguments are required: -p/--
pat
```

```

bash % python3 search.py -v -p spam --pat=eggs foo.txt bar.txt
filenames = ['foo.txt', 'bar.txt']
patterns   = ['spam', 'eggs']
verbose    = True
outfile     = None
speed       = slow

bash % python3 search.py -v -p spam --pat=eggs foo.txt bar.txt
-o results
filenames = ['foo.txt', 'bar.txt']
patterns   = ['spam', 'eggs']
verbose    = True
outfile     = results
speed       = slow

bash % python3 search.py -v -p spam --pat=eggs foo.txt bar.txt
-o results \
    --speed=fast
filenames = ['foo.txt', 'bar.txt']
patterns   = ['spam', 'eggs']
verbose    = True
outfile     = results
speed       = fast

```

```

    print()

```

### 13.3.3

```

argparse

```

```

ArgumentParser
add_argument()

```

`parser.add_argument(dest='filenames',metavar='filename',  
action='store',  
nargs='*')`

`parser.add_argument(dest='verbose',action='store_true',  
help='verbose mode')`

```
parser.add_argument(dest='filenames',metavar='filename',  
nargs='*')
```

`parser.add_argument('-v', dest='verbose', action='store_true',  
help='verbose mode')`

```
parser.add_argument('-v', dest='verbose', action='store_true',  
help='verbose mode')
```

`parser.add_argument('-o', dest='outfile', action='store',  
help='output file')`

```
parser.add_argument('-o', dest='outfile', action='store',  
help='output file')
```

`parser.add_argument('-p', '--pat',metavar='pattern',  
required=True,  
dest='patterns', action='append',`

```
parser.add_argument('-p', '--pat',metavar='pattern',  
required=True,  
dest='patterns', action='append',
```



```
help='text pattern to search for')
```

```
    parser.add_argument('-s', dest='search', action='store',
                        help='text pattern to search for')
    parser.add_argument('-v', dest='verbose', action='store_true',
                        help='verbose output')
```

```
parser.add_argument('--speed', dest='speed', action='store',
                    choices={'slow', 'fast'},
                    default='slow',
                    help='search speed')
```

```
    args = parser.parse_args(sys.argv[1:])
    search = args.search
    verbose = args.verbose
    speed = args.speed
```

```
    # Parse the command line arguments
    sys.argv[1:] = sys.argv[1:]
    getopt = getopt.getopt(sys.argv[1:], 's:v', ['--speed='])
    (opts, args) = getopt.getopt(sys.argv[1:], 's:v', ['--speed='])
    optparse = optparse.OptionParser()
    optparse.add_argument('-s', dest='search', action='store',
                          help='text pattern to search for')
    optparse.add_argument('-v', dest='verbose', action='store_true',
                          help='verbose output')
    optparse.add_argument('--speed', dest='speed', action='store',
                          choices={'slow', 'fast'},
                          default='slow',
                          help='search speed')
```

## 13.4 13.4.1

### 13.4.1

Python 3.6.5 Shell  
Python 3.6.5 Shell  
Python 3.6.5 Shell

## 13.4.2 Python

Python getpass  
Python getpass  
Python getpass

```
import getpass

user = getpass.getuser()
passwd = getpass.getpass()

if
    svc_login(user, passwd): # You must write svc_login()

    print
    ('Yay!')
else
:
    print
    ('Boo!')
```

svc\_login()은 사용자 이름과 비밀번호를  
입력받아 로그인 여부를 판단해주는 함수

### 13.4.3 실행

getpass.getuser()은 사용자 이름  
shell을 실행하고 shell에서 pwd를  
실행하면 현재 디렉토리 경로를 출력

사용자 이름을 입력하고 shell을 실행하고  
pwd를 실행하면 현재 디렉토리 경로를 출력

```
user = input('Enter your username: ')
```

getpass()은 사용자 이름을 입력하고  
Python에서 실행하면 현재 디렉토리  
경로를 출력

## 13.5 파일 처리

### 13.5.1 실행

파일 처리를 위한 모듈은 os와 shutil

## 13.5.2 终端大小

使用 `os.get_terminal_size()` 获取终端大小

```
>>> import os

>>> sz = os.get_terminal_size()
>>> sz
os.terminal_size(columns=80, lines=24)
>>> sz.columns
80
>>> sz.lines
24
>>>
```

## 13.5.3 终端控制

使用 `os.system()` 执行系统命令，`os.system()` 返回的是命令执行的退出状态码。使用 `os.popen()` 打开一个管道，`os.popen()` 返回的是一个文件对象，可以通过该对象向管道写入数据或从管道读取数据。使用 `os.spawnl()` 以单进程方式启动一个新进程，`os.spawnl()` 返回的是新进程的 PID。使用 `os.spawnvp()` 以单进程方式启动一个新进程，`os.spawnvp()` 返回的是新进程的 PID。使用 `os.spawnlpe()` 以单进程方式启动一个新进程，`os.spawnlpe()` 返回的是新进程的 PID。使用 `os.spawnvpe()` 以单进程方式启动一个新进程，`os.spawnvpe()` 返回的是新进程的 PID。

## 13.6 进程间通信

### 13.6.1 管道

管道是一种进程间通信的方式，使用 `os.mkfifo()` 创建管道，`os.mkfifo()` 返回的是管道的名称。使用 `os.open()` 打开管道，`os.open()` 返回的是管道的文件描述符。使用 `os.write()` 向管道写入数据，`os.write()` 返回的是写入的字节数。使用 `os.read()` 从管道读取数据，`os.read()` 返回的是读取的字节数。

## 13.6.2 subprocess

subprocess.check\_output() returns a bytes object

```
import subprocess
```

```
out_bytes = subprocess.check_output(['netstat', '-a'])
```

subprocess.check\_output(['netstat', '-a']) returns a bytes object

```
out_text = out_bytes.decode('utf-8')
```

subprocess.check\_output(['cmd', 'arg1', 'arg2']) returns a bytes object

```
try
:
    out_bytes = subprocess.check_output(['cmd', 'arg1', 'arg2'])
except
subprocess.CalledProcessError as
e:
    out_bytes = e.output          # Output generated before
```

```
error

code = e.returncode           # Return code
```

```
code = e.returncode           # Return code
```

```

    check_output()
    stderr

```

[illegible][illegible]

```
try
:
    out_bytes = subprocess.check_output(['cmd', 'arg1', 'arg2'],
    timeout=5)
except
subprocess.TimeoutExpired as
e:
    ...
```

```

      □□□□□□□□□□□□□□□□shell□□□□□sh□
bash□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```

`os.execve()` 실행 시 `shell` 옵션을 사용하면  
실행된 프로세스가 `shell=True` 옵션을  
Python 인터프리터 I/O 인터페이스를 `shell`  
실행 시 실행된 프로세스가

```
out_bytes = subprocess.check_output('grep python | wc > out',  
shell=True)
```

`shell` 옵션을 사용하면  
실행된 프로세스가 `shlex.quote()` 옵션  
실행 시 실행된 프로세스가 `shell` 옵션을

### 13.6.3

`check_output()` 옵션을 사용하면  
실행된 프로세스가 `subprocess.Popen` 옵션을

```
import subprocess
```

```
# Some text to send
```

```
text = b'''  
hello world  
this is a test  
goodbye
```

```

...

# Launch a command with pipes

p = subprocess.Popen(['wc'],
                      stdout = subprocess.PIPE,
                      stdin = subprocess.PIPE)

# Send the data and get the output

stdout, stderr = p.communicate(text)

# To interpret as text, decode

out = stdout.decode('utf-8')
err = stderr.decode('utf-8')

```

在代码中，我们使用了 `subprocess.Popen` 来启动一个命令，并指定了 `stdout` 和 `stdin` 为 `PIPE`。然后，我们使用 `p.communicate(text)` 来发送数据并获取输出。最后，我们使用 `decode('utf-8')` 来解码输出，使其成为文本。

## 13.7 进程管理

### 13.7.1 进程

在代码中，我们使用了 `subprocess.Popen` 来启动一个命令，并指定了 `stdout` 和 `stdin` 为 `PIPE`。然后，我们使用 `p.communicate(text)` 来发送数据并获取输出。最后，我们使用 `decode('utf-8')` 来解码输出，使其成为文本。



## 13.7.2 □□□□

```
shutil.rmtree(
    os.path.join(
        os.path.dirname(
            os.path.abspath(
                __file__
            )
        ),
        "data"
    )
)
```

```
import shutil

# Copy src to dst. (cp src dst)

shutil.copy(src, dst)

# Copy files, but preserve metadata (cp -p src dst)

shutil.copy2(src, dst)

# Copy directory tree (cp -R src dst)

shutil.copytree(src, dst)

# Move src to dst (mv src dst)

shutil.move(src, dst)
```

follow\_symlinks

```
shutil.copy2(src, dst, follow_symlinks=False)
```

```
shutil.copytree(src, dst, symlinks=True)
```

copytree() ignore

```
def
ignore_pyc_files(dirname, filenames):
    return
[name in
filenames if
name.endswith('.pyc')]
shutil.copytree(src, dst, ignore=ignore_pyc_files)
```

ignore\_patterns()

```
shutil.copytree(src, dst,  
ignore=shutil.ignore_patterns('*~', '*.pyc'))
```

## 13.7.3 路径

Python 的 `shutil` 模块提供了对文件系统的操作。它包含了一个 `copy2()` 函数，它复制文件并保留文件的元数据。它还有一个 `resource forks` 属性，它用于在 Mac OS 上复制资源叉。此外，还有一个 `shutil.copytree()` 函数，它用于复制整个目录树。

Python 的 `os.path` 模块提供了对文件路径的操作。它包含了一些函数，用于处理路径，例如 `os.path.basename()`、`os.path.dirname()`、`os.path.split()`、`os.path.join()` 和 `os.path.expanduser()`。这些函数在 UNIX 和 Windows 上都能使用。

```
>>> filename = '/Users/guido/programs/spam.py'  
>>> import os.path  
  
>>> os.path.basename(filename)  
'spam.py'  
>>> os.path.dirname(filename)  
'/Users/guido/programs'  
>>> os.path.split(filename)  
( '/Users/guido/programs', 'spam.py' )  
>>> os.path.join('/new/dir', os.path.basename(filename))  
'/new/dir/spam.py'  
>>> os.path.expanduser('~ /guido/programs/spam.py')  
'/Users/guido/programs/spam.py'  
>>>
```

`copytree()` 函数用于递归复制文件和目录。它接受两个参数：源路径和目的路径。如果源路径是一个文件，则将其复制到目的路径。如果源路径是一个目录，则递归复制该目录下的所有文件和子目录。该函数在遇到权限错误或目标文件已存在时会抛出异常。

```
try
:
    shutil.copytree(src, dst)
except
shutil.Error as
e:
    for
src, dst, msg in
e.args[0]:
    # src is source name

    # dst is destination name

    # msg is error message from exception

    print
(dst, src, msg)
```

```
ignore_dangling_sumlinks=True
copytree()
```

shutil

<http://docs.python.org/3/library/shutil.html>

## 13.8

### 13.8.1

.tar .tgz .zip

### 13.8.2

shutil — make\_archive() unpack\_archive()

```
>>> import shutil

>>> shutil.unpack_archive('Python-3.3.0.tgz')
>>> shutil.make_archive('py33', 'zip', 'Python-3.3.0')
'/Users/beazley/Downloads/py33.zip'
>>>
```

`make_archive()` 函数使用 `get_archive_formats()` 函数返回的格式列表。

```
>>> shutil.get_archive_formats()
[('bztar', 'bzip2'ed tar-file'), ('gztar', 'gzip'ed tar-
file'),
('tar', 'uncompressed tar file'), ('zip', 'ZIP file')]
>>>
```

### 13.8.3 归档

Python 提供了 `tarfile`、`zipfile`、`gzip`、`bz2` 模块，用于创建和读取归档文件。此外，`shutil` 模块提供了 `make_archive()` 函数，用于创建归档文件。

使用 `shutil` 模块的 `make_archive()` 函数可以创建归档文件。

## 13.9 文件操作

### 13.9.1 文件

os.walk() Python shell

## 13.9.2

os.walk()

```
#!/usr/bin/env python3.3

import os

def
findfile(start, name):
    for
relpath, dirs, files in
os.walk(start):
    if
name in
files:
        full_path = os.path.join(start, relpath, name)
        print
(os.path.normpath(os.path.abspath(full_path)))
if
```

```
__name__ == '__main__':  
    findfile(sys.argv[1], sys.argv[2])
```

```

    findfile.py

```

```
bash % ./findfile.py . myfile.txt
```

### 13.9.3

```
os.walk()
3
```

```

    file=os.path.join(
        './foo//bar',
        os.path.abspath(
            os.path.normpath(

```

[illegible]



```
#!/usr/bin/env python3.3
```

```
import os
```

```
import time
```

```
def
```

```
modified_within(top, seconds):
```

```
    now = time.time()
```

```
    for
```

```
path, dirs, files in
```

```
os.walk(top):
```

```
    for
```

```
name in
```

```
files:
```

```
        fullpath = os.path.join(path, name)
```

```
        if
```

```
os.path.exists(fullpath):
```

```
        mtime = os.path.getmtime(fullpath)
```

```
        if
```

```
mtime > (now - seconds):
```

```
            print
```

```
(fullpath)
```

```
if
```

```
__name__ == '__main__':
```

```
    import sys
```

```
    if
```

```

len(sys.argv) != 3:
    print

('Usage: {} dir seconds'.format(sys.argv[0]))
    raise SystemExit

(1)

modified_within(sys.argv[1], float(sys.argv[2]))

```

os.path.glob
 5.11 to 5.13

## 13.10 configparser

### 13.10.1 configparser

configparser

### 13.10.2 configparser

configparser

```

; config.ini
; Sample configuration file

[installation]

```



```
=====
Brought to you by the Python Cookbook
=====
>>>
```

cfg.write()  
[installation]

```
>>> cfg.set('server','port','9000')
>>> cfg.set('debug','log_errors','False')
>>> import sys

>>> cfg.write(sys.stdout)
[installation]
library = %(prefix)s/lib
include = %(prefix)s/include
bin = %(prefix)s/bin
prefix = /usr/local

[debug]
log_errors = False
show_warnings = False

[server]
port = 9000
nworkers = 32
pid-file = /tmp/spam.pid
root = /www/root
signature =
=====
Brought to you by the Python Cookbook
=====
>>>
```

## 13.10.3

安装时，可以指定安装选项，如安装位置、是否安装调试信息等。  
安装选项包括：  
- "installation"：安装位置，默认为 /usr/local。  
- "debug"：是否安装调试信息，默认为否。  
- "server"：是否安装服务器端，默认为否。

安装 Python 时，可以指定安装选项，如安装位置、是否安装调试信息等。  
安装选项包括：  
- "prefix"：安装位置，默认为 /usr/local。  
- "debug"：是否安装调试信息，默认为否。  
- "server"：是否安装服务器端，默认为否。

```
prefix=/usr/local  
prefix: /usr/local
```

安装 Python 时，可以指定安装选项，如安装位置、是否安装调试信息等。

```
>>> cfg.get('installation','PREFIX')  
'/usr/local'  
>>> cfg.get('installation','prefix')  
'/usr/local'  
>>>
```

安装 Python 时，可以指定安装选项，如安装位置、是否安装调试信息等。  
安装选项包括：  
- "getboolean"：获取布尔值，默认为否。

```
log_errors = true  
log_errors = TRUE  
log_errors = Yes  
log_errors = 1
```

Python  
prefix

```
[installation]
library=%(prefix)s/lib
include=%(prefix)s/include
bin=%(prefix)s/bin
prefix=/usr/local
```

ConfigParser

```
; ~/.config.ini
[installation]
prefix=/Users/beazley/test

[debug]
log_errors=False
```

```
>>> # Previously read configuration

>>> cfg.get('installation', 'prefix')
'/usr/local'

>>> # Merge in user-specific configuration
```

```
>>> import os

>>> cfg.read(os.path.expanduser('~/.config.ini'))
['/Users/beazley/.config.ini']
>>> cfg.get('installation', 'prefix')
'/Users/beazley/test'
>>> cfg.get('installation', 'library')
'/Users/beazley/test/lib'
>>> cfg.getboolean('debug', 'log_errors')
False
>>>
```

prefix  
library

```
>>> cfg.get('installation', 'library')
'/Users/beazley/test/lib'
>>> cfg.set('installation', 'prefix', '/tmp/dir')
>>> cfg.get('installation', 'library')
'/tmp/dir/lib'
>>>
```

Python  
Windows  
configparser

## 13.11

### 13.11.1

### 13.11.2 □□□□

```
import logging

def
main():
    # Configure the logging system

    logging.basicConfig(
        filename='app.log',
        level=logging.ERROR
    )

    # Variables (to make the calls that follow work)

    hostname = 'www.python.org'
    item = 'spam'
    filename = 'data.csv'
    mode = 'r'
    # Example logging calls (insert into your program)

    logging.critical('Host %s unknown', hostname)
    logging.error("Couldn't find %r", item)
    logging.warning('Feature is deprecated')
    logging.info('Opening file %r, mode=%r', filename, mode)
    logging.debug('Got here')

if
```



```
__name__ == '__main__':  
    main()
```

```

5 logging.critical()
error()
warning()
info()
debug()
basicConfig()
level

```

[illegible]

```
□□□□□□□□□□ app.log □□□□□□□□□□
```

```
CRITICAL:root:Host www.python.org unknown
ERROR:root:Could not find 'spam'
```

```
basicConfig()
```

```
logging.basicConfig(
    filename='app.log',
    level=logging.WARNING,
    format='%(levelname)s:%(asctime)s:%(message)s')
```

[illegible]







```
# Example function (for testing)
```

def

```
func():
    log.critical('A Critical Error!')
    log.debug('A debug message')
```

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

```
>>> import somelib
```

```
>>> somelib.func()
```

&gt;&gt;&gt;

[illegible]

11

```
>>> import logging
```

```
>>> logging.basicConfig()
```

```
>>> somelib.func()
```

CRITICAL:somelib:A Critical Error!

>>>

## 13.12.3 测试

```
import logging
logging.basicConfig(level=logging.ERROR)
import somelib
```

```
logger = logging.getLogger(__name__)
logger.error('A Critical Error!')
```

```
log.addHandler(logging.NullHandler())
import logging
import somelib
logger = logging.getLogger(__name__)
```

```
logger.error('A Critical Error!')
```

```
>>> import logging

>>> logging.basicConfig(level=logging.ERROR)
>>> import somelib

>>> somelib.func()
CRITICAL:somelib:A Critical Error!

>>> # Change the logging level for 'somelib' only

>>> logging.getLogger('somelib').level=logging.DEBUG
```

```
>>> somelib.func()  
CRITICAL:somelib:A Critical Error!  
DEBUG:somelib:A debug message  
>>>
```

□□□□□□□□□□□□□□□□**ERROR**□□□□□□□□□  
□□□somelib□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

“Logging  
HOWTO”□[http://docs.python.org/3/howto/  
logging.html](http://docs.python.org/3/howto/logging.html) □□□□□□□□□□logging□□□□□□□□□□  
□□□□□□□□□□

## **13.13** □□□□□□□□□□

### **13.13.1** □□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□

### **13.13.2** □□□□







```

    while
n > 0:
    n -= 1

# Use 1: Explicit start/stop

t = Timer()
t.start()
countdown(1000000)
t.stop()
print

(t.elapsed)

# Use 2: As a context manager

with

t:
    countdown(1000000)
print

(t.elapsed)

with

Timer() as

t2:
    countdown(1000000)
print

(t2.elapsed)

```

## 13.13.3

time.time() 和 time.clock() 在 Windows 上不可用，  
因此我们使用 time.perf\_counter() 来测量时间。

time.time() 和 time.clock() 在 Windows 上不可用，  
因此我们使用 time.perf\_counter() 来测量时间。

time.perf\_counter() 和 time.process\_time() 都是测量 CPU 时间。  
time.process\_time() 测量的是用户 CPU 时间。

```
t = Timer(time.process_time)
with
t:
    countdown(1000000)
print
(t.elapsed)
```

time.perf\_counter() 和 time.process\_time() 都是测量 CPU 时间。  
time.process\_time() 测量的是用户 CPU 时间。

time.perf\_counter() 和 time.process\_time() 都是测量 CPU 时间。  
time.process\_time() 测量的是用户 CPU 时间。

## 13.14 `resource` CPU 限制

### 13.14.1 简介

`resource` 模块是 UNIX 系统上限制 CPU 使用量的接口。

### 13.14.2 使用

`resource` 模块提供了限制 CPU 使用量的接口。

```
import signal

import resource

import os

def
time_exceeded(signo, frame):
    print
    ("Time's up!")
    raise SystemExit

(1)

def
```

```

set_max_runtime(seconds):
    # Install the signal handler and set a resource limit

    soft, hard = resource.getrlimit(resource.RLIMIT_CPU)
    resource.setrlimit(resource.RLIMIT_CPU, (seconds, hard))
    signal.signal(signal.SIGXCPU, time_exceeded)

if
__name__ == '__main__':
    set_max_runtime(15)
    while
True:
        pass

```

SIGXCPU

```

import resource

def
limit_memory(maxsize):
    soft, hard = resource.getrlimit(resource.RLIMIT_AS)
    resource.setrlimit(resource.RLIMIT_AS, (maxsize, hard))

```

MemoryError

### 13.14.3

setrlimit()

setrlimit() resource

UNIX Linux OS X

## 13.15 Web

### 13.15.1

URL

### 13.15.2

# webbrowser

```
>>> import webbrowser

>>> webbrowser.open('http://www.python.org')
True
>>>
```

```
>>> # Open the page in a new browser window

>>> webbrowser.open_new('http://www.python.org')
True
>>>

>>> # Open the page in a new browser tab

>>> webbrowser.open_new_tab('http://www.python.org')
True
>>>
```

webbrowser.get() Python 3.0

```
>>> c = webbrowser.get('firefox')
>>> c.open('http://www.python.org')
True
>>> c.open_new_tab('http://docs.python.org')
True
>>>
```

Python 3.0  
<http://docs.python.org/3/library/webbrowser.html>

## 13.15.3

Python 3.0  
webbrowser module  
HTML  
webbrowser module



## 14 测试驱动开发

测试驱动开发（Test-Driven Development，TDD）是一种软件开发方法，它要求开发者在编写代码之前，先编写测试用例。这种方法可以确保代码的正确性，并且可以及时发现和修复错误。Python 是一种非常适合 TDD 的编程语言，因为它具有简洁的语法和强大的测试框架。在本章中，我们将介绍 TDD 的基本概念，并演示如何使用 Python 的 unittest 框架进行单元测试。

### 14.1 测试 stdout

#### 14.1.1 测试

在测试驱动开发中，测试是核心。在 Python 中，我们可以使用 `sys.stdout` 来测试程序的输出。通过捕获 `sys.stdout` 的输出，我们可以验证程序是否按照预期工作。以下是一个简单的示例，演示如何测试一个程序的输出。

#### 14.1.2 测试

在 Python 中，我们可以使用 `unittest.mock.patch()` 来模拟 `sys.stdout` 的输出。通过模拟输出，我们可以验证程序是否按照预期工作。以下是一个简单的示例，演示如何使用 `unittest.mock` 进行测试。

在代码中，我们使用 `mymodule` 来模拟输出。

```
# mymodule.py

def urlprint(protocol, host, domain):
    url = '{}://{}.{}'.format(protocol, host, domain)
    print(url)
```

```
import sys
import sys.stdout
sys.stdout = StringIO()
unittest.mock.patch('sys.stdout', new=StringIO())
mymodule.urlprint('http', 'www', 'example.com')
```

```
from io import StringIO
from unittest import TestCase
from unittest.mock import patch
import mymodule

class TestURLPrint(TestCase):
    def test_url_gets_to_stdout(self):
        protocol = 'http'
        host = 'www'
        domain = 'example.com'
        expected_url = '{}://{}.{}\n'.format(protocol, host,
        domain)

        with patch('sys.stdout', new=StringIO()) as fake_out:
            mymodule.urlprint(protocol, host, domain)
            self.assertEqual(fake_out.getvalue(),
            expected_url)
```

## 14.1.3 测试

```
urlprint() dummy value expected_url
```

```
unittest.mock.patch(sys.stdout StringIO fake_out with fake_out with patch()
```

```
C sys.stdout Python C I/O
```

```
StringIO I/O 5.6
```

## 14.2

### 14.2.1

unittest.mock.patch()

## 14.2.2 patch()

unittest.mock.patch() is a decorator that patches the target function with a mock object. The mock object is created by the patch() function and is used to replace the target function during the test. The mock object has an assert\_called\_with() method that can be used to verify that the target function was called with the expected arguments.

```
from unittest.mock import patch
import example

@patch('example.func')
def test1(x, mock_func):
    example.func(x)          # Uses patched example.func
    mock_func.assert_called_with(x)
```

unittest.mock.patch()

```
with patch('example.func') as mock_func:
    example.func(x)          # Uses patched example.func
    mock_func.assert_called_with(x)
```

unittest.mock.patch()

```
p = patch('example.func')
mock_func = p.start()
example.func(x)
mock_func.assert_called_with(x)
p.stop()
```

patch()는 unittest.mock 모듈에서 제공하는 함수로, 모듈의 함수나 클래스를 임시로 대체할 수 있다. patch()는 patcher 객체를 반환하며, patcher 객체는 patch()를 호출한 모듈의 함수나 클래스를 대체한다. patcher 객체는 patch()를 호출한 모듈의 함수나 클래스를 대체한다.

```
@patch('example.func1')
@patch('example.func2')
@patch('example.func3')
def test1(mock1, mock2, mock3):
    ...

def test2():
    with patch('example.patch1') as mock1, \
        patch('example.patch2') as mock2, \
        patch('example.patch3') as mock3:
        ...
```

## 14.2.3 patch()

patch()는 unittest.mock 모듈에서 제공하는 함수로, 모듈의 함수나 클래스를 임시로 대체할 수 있다. patch()는 patcher 객체를 반환하며, patcher 객체는 patch()를 호출한 모듈의 함수나 클래스를 대체한다. patcher 객체는 patch()를 호출한 모듈의 함수나 클래스를 대체한다.

```
>>> x = 42
>>> with patch('__main__.x'):
...     print(x)
...
<MagicMock name='x' id='4314230032'>
>>> x
42
>>>
```

patch()는 unittest.mock 모듈에서 제공하는 함수로, 모듈의 함수나 클래스를 임시로 대체할 수 있다. patch()는 patcher 객체를 반환하며, patcher 객체는 patch()를 호출한 모듈의 함수나 클래스를 대체한다. patcher 객체는 patch()를 호출한 모듈의 함수나 클래스를 대체한다.

```

>>> x
42
>>> with patch('__main__.x', 'patched_value'):
...     print(x)
...
patched_value
>>> x
42
>>>

```

## MagicMock

```

>>> from unittest.mock import MagicMock
>>> m = MagicMock(return_value = 10)
>>> m(1, 2, debug=True)
10
>>> m.assert_called_with(1, 2, debug=True)
>>> m.assert_called_with(1, 2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File ".../unittest/mock.py", line 726, in assert_called_with
    raise AssertionError(msg)
AssertionError: Expected call: mock(1, 2)
Actual call: mock(1, 2, debug=True)
>>>

>>> m.upper.return_value = 'HELLO'
>>> m.upper('hello')
'HELLO'
>>> assert m.upper.called

>>> m.split.return_value = ['hello', 'world']
>>> m.split('hello world')
['hello', 'world']
>>> m.split.assert_called_with('hello world')
>>>

>>> m['blah']
<MagicMock name='mock.__getitem__()' id='4314412048'>
>>> m.__getitem__.called

```

```
True
>>> m.__getitem__.assert_called_with('blah')
>>>
```

```
# example.py
from urllib.request import urlopen
import csv

def dowprices():
    u = urlopen('http://finance.yahoo.com/d/quotes.csv?
s=@^DJI&f=s11')
    lines = (line.decode('utf-8') for line in u)
    rows = (row for row in csv.reader(lines) if len(row) == 2)
    prices = { name:float(price) for name, price in rows }
    return prices
```

```

urlopen() Web

```

```
import unittest
from unittest.mock import patch
import io
import example

sample_data = io.BytesIO(b'''\
"IBM",91.1\r
"AA",13.25\r
"MSFT",27.72\r
\r
''')
```

```

class Tests(unittest.TestCase):
    @patch('example.urlopen', return_value=sample_data)
    def test_dowprices(self, mock_urlopen):
        p = example.dowprices()
        self.assertTrue(mock_urlopen.called)
        self.assertEqual(p,
                          {'IBM': 91.1,
                           'AA': 13.25,
                           'MSFT': 27.72})

if __name__ == '__main__':
    unittest.main()

```

1. 在example模块中定义urlopen()函数  
 2. 使用mock模块中的BytesIO()函数来模拟urlopen()  
 3.

4. 在example模块中定义urlopen()函数  
 5. 在example模块中定义urlopen()函数  
 6. 在example模块中定义urlopen()函数  
 7. 在example模块中定义urlopen()函数  
 8. 在example模块中定义urlopen()函数  
 9. 在example模块中定义urlopen()函数  
 10. 在example模块中定义urlopen()函数

11. 在example模块中定义urlopen()函数  
 12. 在example模块中定义urlopen()函数  
 13. <http://docs.python.org/3/library/unittest.mock>



## 14.3 unittest.TestCase

### 14.3.1 setUp()

unittest.TestCase.setUp()

### 14.3.2 assertRaises()

unittest.TestCase.assertRaises() unittest.TestCase.assertRaises(ValueError)

```
import unittest

# A simple function to illustrate

def
parse_int(s):
    return
int(s)

class TestConversion
(unittest.TestCase):
    def
test_bad_int(self):
        self.assertRaises(ValueError
, parse_int, 'N/A')
```

assertRaises()는 예외가 발생하면 성공하고, 예외가 발생하지 않으면 실패한다. assertRaises()는 try-except 문과 함께 사용된다.

```
import errno

class TestIO
(unittest.TestCase):
    def
test_file_not_found(self):
    try
:
        f = open('/file/not/found')
    except IOError as
e:
        self.assertEqual(e.errno, errno.ENOENT)
    else
:
        self.fail('IOError not raised')
```

### 14.3.3 assertRaises()

assertRaises()는 예외가 발생하면 성공하고, 예외가 발생하지 않으면 실패한다. assertRaises()는 try-except 문과 함께 사용된다.



```
:
    self.fail('ValueError not raised')
```

`assertRaises()` 测试是否抛出了指定的异常

`assertRaises()` 测试是否抛出了指定的异常

`assertRaisesRegex()` 测试是否抛出了指定的异常，并且匹配指定的正则表达式

```
class TestConversion
(unittest.TestCase):
    def
test_bad_int(self):
    self.assertRaisesRegex(ValueError
, 'invalid literal .*',
                           parse_int, 'N/A')
```

`assertRaises()` 测试是否抛出了指定的异常

```
class TestConversion
(unittest.TestCase):
```

```
def
test_bad_int(self):
    with
self.assertRaisesRegex(ValueError
, 'invalid literal .*'):
    r = parse_int('N/A')
```

## 14.4

### 14.4.1 ☐☐

[illegible]

## 14.4.2 ☐☐☐☐

[illegible]

```
import unittest

class MyTest
(unittest.TestCase):
```

```

...

if
__name__ == '__main__':
    unittest.main()

```

unittest.main()
 unittest.main()

```

import sys

def
main(out=sys.stderr, verbosity=2):
    loader = unittest.TestLoader()
    suite = loader.loadTestsFromModule(sys.modules[__name__])
    unittest.TextTestRunner(out,verbosity=verbosity).run(suite)

if
__name__ == '__main__':
    with
open('testing.out', 'w') as
f:
    main(f)

```

## 14.4.3

unittest.TestCase 的 load\_tests 方法，  
返回一个 TestLoader 对象，该对象负责加载测试用例。

unittest.TestCase 的 load\_tests 方法，  
返回一个 TestLoader 对象，该对象负责加载测试用例。  
该对象负责加载测试用例。

unittest.TestCase 的 load\_tests 方法，  
返回一个 TestLoader 对象，该对象负责加载测试用例。  
load\_tests 方法返回一个 TestLoader 对象，  
该对象负责加载测试用例。  
load\_tests 方法返回一个 TestLoader 对象，  
该对象负责加载测试用例。  
load\_tests 方法返回一个 TestLoader 对象，  
该对象负责加载测试用例。

unittest.TestCase 的 load\_tests 方法，  
返回一个 TestLoader 对象，该对象负责加载测试用例。  
unittest.TestCase 的 load\_tests 方法，  
返回一个 TestLoader 对象，该对象负责加载测试用例。  
unittest.TestCase 的 load\_tests 方法，  
返回一个 TestLoader 对象，该对象负责加载测试用例。

unittest.TestCase 的 load\_tests 方法，  
返回一个 TestLoader 对象，该对象负责加载测试用例。  
unittest.TestCase 的 load\_tests 方法，  
返回一个 TestLoader 对象，该对象负责加载测试用例。  
unittest.TestCase 的 load\_tests 方法，  
返回一个 TestLoader 对象，该对象负责加载测试用例。  
unittest.TestCase 的 load\_tests 方法，  
返回一个 TestLoader 对象，该对象负责加载测试用例。

## 14.5 unittest 라이브러리

### 14.5.1 개요

unittest 라이브러리는 Python 2.7부터 도입된 표준 라이브러리이다. unittest 라이브러리를 사용하여 테스트 코드를 작성할 수 있다.

### 14.5.2 unittest 라이브러리

unittest 라이브러리를 사용하여 테스트 코드를 작성할 수 있다. unittest 라이브러리를 사용하여 테스트 코드를 작성할 수 있다.

```
import unittest

import os

import platform

class Tests
(unittest.TestCase):
    def
test_0(self):
    self.assertTrue(True)

    @unittest.skip('skipped test')
    def
test_1(self):
```



```

        self.fail('should have failed!')

    @unittest.skipIf(os.name=='posix', 'Not supported on
Unix')
    def
test_2(self):
    import winreg

    @unittest.skipUnless(platform.system() == 'Darwin', 'Mac
specific test')
    def

test_3(self):
    self.assertTrue(True)

    @unittest.expectedFailure
    def

test_4(self):
    self.assertEqual(2+2, 5)

if
__name__ == '__main__':
    unittest.main()

```

Mac

```

bash % python3 testsample.py -v
test_0 (__main__.Tests) ... ok
test_1 (__main__.Tests) ... skipped 'skipped test'
test_2 (__main__.Tests) ... skipped 'Not supported on Unix'
test_3 (__main__.Tests) ... ok
test_4 (__main__.Tests) ... expected failure

-----
-----
Ran 5 tests in 0.002s

```

```
OK (skipped=2, expected failures=1)
```

## 14.5.3 `unittest`

`unittest.skip()` and `unittest.skipIf()` and `unittest.skipUnless()` are used to skip tests. `unittest.skipUnless()` is used to skip tests on Python 2.5 and earlier. The decorator `@unittest.expectedFailure` is used to mark tests that are expected to fail.

Example: Skipping tests on Darwin and Mac OS X

```
@unittest.skipUnless(platform.system() == 'Darwin', 'Mac
specific tests')
class DarwinTests
(unittest.TestCase):
    ...
```

## 14.6 `unittest`

### 14.6.1 `unittest`

Example: Skipping tests on Darwin and Mac OS X

## 14.6.2 断点续传

断点续传是指从上次断开的地方继续下载，而不需要重新下载整个文件。这通常用于大文件的下载，以提高效率和节省带宽。

```
try
:
    client_obj.get_url(url)
except
(URLError, ValueError
, SocketTimeout):
    client_obj.remove_url(url)
```

remove\_url() 方法用于从断点续传列表中移除指定的 URL。该方法通常与 get\_url() 方法一起使用，以确保在断点续传过程中能够正确地管理和更新下载列表。

```
try
:
    client_obj.get_url(url)
except
(URLError, ValueError
):
    client_obj.remove_url(url)
except
SocketTimeout:
```

```
client_obj.handle_url_timeout(url)
```

try  
: f = open(filename)  
except (FileNotFoundError, PermissionError):  
...

try

:

f = open(filename)

except

(FileNotFoundError, PermissionError):

...

except

try

:

f = open(filename)

except OSError

:

...

OSError  
FileNotFoundError  
PermissionError

## 14.6.3 `try`

`try`은 `except`과 함께 사용되며, `except`은 `try` 블록에서 발생하는 예외를 처리하는 데 사용됩니다. `except`은 `try` 블록의 실행이 성공적으로 완료되면 실행되지 않습니다.

```
try
:
    f = open(filename)
except OSError as e:
    if
e.errno == errno.ENOENT:
        logger.error('File not found')
    elif
e.errno == errno.EACCES:
        logger.error('Permission denied')
    else
:
        logger.error('Unexpected error: %d', e.errno)
```

`except`은 `try` 블록에서 발생하는 예외를 처리하는 데 사용됩니다. `except`은 `try` 블록의 실행이 성공적으로 완료되면 실행되지 않습니다.

`except`은 `try` 블록에서 발생하는 예외를 처리하는 데 사용됩니다. `except`은 `try` 블록의 실행이 성공적으로 완료되면 실행되지 않습니다.

```

>>> f = open('missing')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory:
'missing'
>>> try

:
...

f = open('missing')
... except OSError

:
...     print

('It failed')
... except

FileNotFoundError:
...     print

('File not found')
...

It failed
>>>

```

except FileNotFoundError  
 OSError FileNotFoundError  
 FileNotFoundError

mro

```
>>> FileNotFoundError.__mro__
(<class 'FileNotFoundError'>, <class 'OSError'>, <class
'Exception'>,
<class 'BaseException'>, <class 'object'>)
>>>
```

BaseException

except

## 14.7

### 14.7.1

### 14.7.2

Exception

```
try
:
...
except Exception as
e:
...
    log('Reason:', e)    # Important!
```

`SystemExit` `KeyboardInterrupt`  
`GeneratorExit` `Exception` `BaseException`

### 14.7.3

```
def
parse_int(s):
    try
:
        n = int(v)
    except Exception
:
        print
```



```
("Couldn't parse")
```

```
□□□□□□□□□□□□□□□□□□□□
```

```
>>> parse_int('n/a')
Couldn't parse
>>> parse_int('42')
Couldn't parse
>>>
```

```
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□
```

```
def
parse_int(s):
    try
:
        n = int(v)
    except Exception as
e:
        print
("Couldn't parse")
        print
('Reason:', e)
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□

```
>>> parse_int('42')
Couldn't parse
Reason: global name 'v' is not defined
>>>
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□

# 14.8 □□□□□□□□

## 14.8.1 □□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□

## 14.8.2 □□□□

□□□□□□□□□□□□——□□□□□□□□□□□□  
Exception□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□

```
class NetworkError
```

```
(Exception
```

```
):
```

```
    pass
```

```
class HostnameError
```

```
(NetworkError):
```

```
    pass
```

```
class TimeoutError
```

```
(NetworkError):
```

```
    pass
```

```
class ProtocolError
```

```
(NetworkError):
```

```
    pass
```

```
□□□□□□□□□□□□□□□□□□□□□□□□
```

```
try
```

```
:
```

```
    msg = s.recv()
```

```
except
```

```
TimeoutError as
```

```

e:
    ...
except
    ProtocolError as
e:
    ...

```

## 14.8.3 异常

异常是程序运行过程中出现的错误，通常由异常类（Exception）表示。异常类是 Python 中 Exception 类的一个子类，它继承自 Exception 类。BaseException 是 Python 中异常类的基类，所有异常类都必须继承自 BaseException。KeyboardInterrupt 和 SystemExit 是 Python 中两个特殊的异常类，它们分别表示键盘中断和系统退出。

异常处理是 Python 中一个重要的概念，它允许程序员在程序运行过程中捕获并处理异常。Python 提供了 try、except、finally 等关键字来实现异常处理。try 语句用于包裹可能引发异常的代码块，except 语句用于捕获并处理异常，finally 语句用于在异常处理完成后执行的代码块。

```

try
:
    s.send(msg)

```

**except**

```
ProtocolError:
```

■ ■ ■

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

```
try
```

□  
□

```
s.send(msg)
```

**except**

NetworkError:

■ ■ ■

```

    def __init__(self):
        Exception.__init__()

```

```
class CustomError
```

**(Exception**

$$):$$
**def**

```
__init__(self, message, status):
    super().__init__(message, status)
    self.message = message
    self.status = status
```



Python  
[http://docs.python.org/3/tutorial/errors.html](http://docs.python.org/3/tutorial/errors.html)

## 14.9

### 14.9.1

traceback

### 14.9.2

raise from  
raise

```
>>> def
example():
...     try
:
...
int('N/A')
...     except ValueError as
```





```
if
e.__cause__:
    print
('Cause:', e.__cause__)
```

except

```
>>> def
example2():
...     try
:
...
int('N/A')
...     except ValueError as
e:
...         print
('Couldn't parse:', err)
...

>>>
>>> example2()
Traceback (most recent call last):
  File "<stdin>", line 3, in example2
ValueError: invalid literal for int() with base 10: 'N/A'

During handling of the above exception, another exception
occurred:

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 5, in example2
```

```
NameError: global name 'err' is not defined
>>>
```

```

    """
    """
    """NameError"""
    """
    """cause """
    """context """
ValueError"""
```

```

    """raise from
None"""
```

```
>>> def
example3():
...     try
:
...
int('N/A')
...     except ValueError
:
...         raise RuntimeError
('A parsing error occurred') from None...
...

>>> example3()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 5, in example3
```

```
RuntimeError: A parsing error occurred
>>>
```

### 14.9.3 ☐☐

```

except:
    raise
    raise
from

```

```
try
:
...
except
SomeException as
e:
    raise
DifferentException() from e
```

```

Different ExceptionSomeException
traceback

```

try  
except

```
try
:
...
except
SomeException:
    raise
DifferentException()
```

raise from

traceback

## 14.10

### 14.10.1

except

## 14.10.2 `raise`

`raise` raises an exception

```
>>> def
example():
...     try
:
...
int('N/A')
...     except ValueError
:
...         print
("Didn't work")
...         raise
...
>>> example()
Didn't work
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in example
ValueError: invalid literal for int() with base 10: 'N/A'
>>>
```

## 14.10.3 `try`

`try` is used to catch exceptions



warnings.warn()  
warnings.warn()

```
import warnings

def
func(x, y, logfile=None, debug=False):
    if
logfile is not
None:
        warnings.warn('logfile argument deprecated',
        DeprecationWarning
    )
    ...
```

warn()  
UserWarning  
DeprecationWarning  
SyntaxWarning  
RuntimeWarning  
ResourceWarning  
FutureWarning

-W all  
Python

```
bash % python3 -W all example.py
example.py:5: DeprecationWarning: logfile argument is
deprecated
    warnings.warn('logfile argument is deprecated',
```

```
DeprecationWarning)
```

python3 -W error example.py

```
bash % python3 -W error example.py
Traceback (most recent call last):
  File "example.py", line 10, in <module>
    func(2, 3, logfile='log.txt')
  File "example.py", line 5, in func
    warnings.warn('logfile argument is deprecated',
DeprecationWarning)
DeprecationWarning: logfile argument is deprecated
bash %
```

## 14.11.3 警告

警告是Python解释器在遇到某些可能引发错误的行为时发出的消息。警告通常是由于使用了过时的API或语法。警告消息通常以DeprecationWarning或FutureWarning的形式出现。警告消息可以帮助开发者了解代码中的潜在问题，并及时进行修复。

警告消息通常以warning模块中的Warning类为基类。警告消息的格式通常为：Warning: message。警告消息可以在代码中通过warnings.warn()函数发出。

```
>>> import warnings

>>> warnings.simplefilter('always')
>>> f = open('/etc/passwd')
```



```
>>> del

f
__main__:1: ResourceWarning: unclosed file <_io.TextIOWrapper
name='/etc/passwd'
mode='r' encoding='UTF-8'>
>>>
```

warnings.simplefilter("always")  
 warnings.simplefilter("ignore", category=ResourceWarning)

warnings.simplefilter("always")  
 warnings.simplefilter("ignore", category=ResourceWarning)

Python  
<http://docs.python.org/3/library/warnings.html>

## 14.12 警告过滤器

### 14.12.1 警告过滤器

warnings.simplefilter("always")

## 14.12.2 실행

실행하려면 파이썬 인터프리터를 실행하고 `python3 -i someprogram.py`를 입력하면 됩니다. 이 명령어는 셸(shell)을 실행하고, `someprogram.py` 파일을 실행합니다.

```
# sample.py

def
func(n):
    return
n + 10
func('Hello')
```

`python3 -i`를 실행하면

```
bash % python3 -i sample.py
Traceback (most recent call last):
  File "sample.py", line 6, in <module>
    func('Hello')
  File "sample.py", line 4, in func
    return
n + 10
TypeError: Can't convert 'int' object to str implicitly
>>> func(10)
20
>>>
```

Python

```
>>> import pdb

>>> pdb.pm()
> sample.py(4)func()
-> return n + 10
(Pdb) w
sample.py(6)<module>()
-> func('Hello')
> sample.py(4)func()
-> return n + 10
(Pdb) print n
'Hello'
(Pdb) q
>>>
```

shell

traceback

```
import traceback

import sys

try
:
    func(arg)
```

```
except
:
    print
('**** AN ERROR OCCURRED ****')
    traceback.print_exc(file=sys.stderr)
```

```

    print()
    traceback.print_stack()

```

```
>>> def
sample(n):
...     if
n > 0:
...
sample(n-1)
...     else
:
...
traceback.print_stack(file=sys.stderr)
...

>>> sample(5)
File "<stdin>", line 1, in <module>
File "<stdin>", line 3, in sample
File "<stdin>", line 3, in sample
File "<stdin>", line 3, in sample
File "<stdin>", line 3, in sample
```

```
File "<stdin>", line 3, in sample
File "<stdin>", line 5, in sample
>>>
```

pdb.set\_trace()

```
import pdb

def
func(arg):
    ...
    pdb.set_trace()
    ...
```

print

### 14.12.3

traceback

```
print()

```

```


```

```

pdb.set_trace()
set_trace()

```

```

IDE Python IDE
pdb pdb
IDE

```

## 14.13

### 14.13.1

```


```

### 14.13.2

# time on UNIX

```
bash % time python3 someprogram.py
real 0m13.937s
user 0m12.162s
sys   0m0.098s
bash %
```

# cProfile

```
bash % python3 -m cProfile someprogram.py
      859647 function calls in 16.016 CPU seconds

Ordered by: standard name

ncalls  tottime  percall  cumtime  percall
filename:lineno(function)
263169  0.080    0.000    0.080    0.000
someprogram.py:16(frange)
513    0.001    0.000    0.002    0.000
someprogram.py:30(generate_mandel)
262656  0.194    0.000    15.295    0.000
someprogram.py:32(<genexpr>)
1      0.036    0.036    16.077    16.077
someprogram.py:4(<module>)
262144  15.021    0.000    15.021    0.000
someprogram.py:4(in_mandelbrot)
1      0.000    0.000    0.000    0.000    os.py:746(urandom)
1      0.000    0.000    0.000    0.000
png.py:1056(_readable)
1      0.000    0.000    0.000    0.000    png.py:1073(Reader)
1      0.227    0.227    0.438    0.438    png.py:163(<module>)
512    0.010    0.000    0.010    0.000    png.py:200(group)
...
bash %
```

```
# timethis.py

import time

from functools import wraps

def timethis(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        start = time.perf_counter()
        r = func(*args, **kwargs)
        end = time.perf_counter()
        print('{}.{} : {}'.format(func.__module__, func.__name__, end - start))
        return r
    return wrapper
```



□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□

```
>>> @timethis
... def
countdown(n):
...     while
n > 0:
...
n -= 1
...

>>> countdown(10000000)
__main__.countdown : 0.803001880645752
>>>
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□

```
from contextlib import
contextmanager

@contextmanager
def
timeblock(label):
    start = time.perf_counter()
    try
:
        yield
```

```

finally
:
    end = time.perf_counter()
    print
('{} : {}'.format(label, end - start))

```

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

```
>>> with
timeblock('counting'):
...
n = 100000000
...     while
n > 0:
...
n -= 1
...
counting : 1.5551159381866455
>>>
```

```

timeit

```

```
>>> from timeit import
timeit
>>> timeit('math.sqrt(2)', 'import math')
```

```
0.1432319980012835
>>> timeit('sqrt(2)', 'from math import sqrt')
0.10836604500218527
>>>
```

`timeit` 模块提供了 `timeit` 函数，用于测量代码片段的执行时间。该函数接受两个参数：要测量的代码片段（字符串）和要导入的模块名称。此外，还可以指定要执行的次数（默认为 1000000）。

```
>>> timeit('math.sqrt(2)', 'import math', number=10000000)
1.434852126003534
>>> timeit('sqrt(2)', 'from math import sqrt',
number=10000000)
1.0270336690009572
>>>
```

## 14.13.3 性能测试

Python 提供了多种性能测试工具。其中，`time.perf_counter()` 用于测量 CPU 时间，而 `time.time()` 用于测量 wall-clock time（即从系统启动到现在的总时间）。此外，还有 `time.process_time()` 用于测量进程时间。

`time.process_time()` 用于测量进程的 CPU 时间，包括用户时间和系统时间。该函数返回一个浮点数，表示从系统启动到现在的总时间。

```
from functools import
wraps
```

```

def
timethis(func):
    @wraps(func)
    def
wrapper(*args, **kwargs):
    start = time.process_time()
    r = func(*args, **kwargs)
    end = time.process_time()
    print
('{}.{}} : {}'.format(func.__module__, func.__name__, end -
start))
    return
r
    return
wrapper

```

timeit

13.13

## 14.14

### 14.14.1

**编译时C++JIT**

## 14.14.2

[illegible]

“hotspot”

□ □ □ □

# Python

```
# somescript.py
```

```
import sys
```

```
import csv
```

**with**

```
open(sys.argv[1]) as
f:
    for
row in
csv.reader(f):
    # Some kind of processing

    ...
```

```
# somescript.py

import sys

import csv

def
main(filename):
    with
open(filename) as
f:
    for
```

```
row in
csv.reader(f):
    # Some kind of processing

    ...

main(sys.argv[1])
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
15%□30%□□□□□□□□

□□□□□□□□□□□□

□□□□□□□□□□.□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□**getattr** ()□**getattr** ()□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□from module import name□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
import math

def
compute_roots(nums):
    result = []
```

```

        for
n in
nums:
    result.append(math.sqrt(n))
    return
result
# Test

nums = range(1000000)
for
n in
range(100):
    r = compute_roots(nums)

```

40
 compute\_roots()

```

from math import
sqrt
def
compute_roots(nums):
    result = []
    result_append = result.append
    for
n in
nums:
    result_append(sqrt(n))

```



```
return  
result
```

```
    # Compute the square root of 29 using math.sqrt()  
    # Append the result to the result list  
    result.append(math.sqrt(29))  
    # Append the result to the result list  
    result.append(29)
```

```
    # Compute the square root of 29 using math.sqrt()  
    # Append the result to the result list  
    result.append(math.sqrt(29))
```

```
    # Compute the square root of 29 using math.sqrt()  
    # Append the result to the result list
```

```
    # Compute the square root of 29 using math.sqrt()  
    # Append the result to the result list  
    result.append(math.sqrt(29))  
    # Append the result to the result list  
    result.append(29)
```

```
import math  
  
def  
compute_roots(nums):  
    sqrt = math.sqrt  
    result = []  
    result_append = result.append  
    for  
n in
```

```
nums:
    result_append(sqrt(n))
    return
result
```

□□□□□□sqrt□□□□□math□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□25□□□□□□□  
□□29□□□□□□□□□□□□□□□□□□□□□□sqrt□□□□□□  
□□□sqrt□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
self.name□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
# Slower

class SomeClass
:
    ...
    def
method(self):
    for
x in
s:
    op(self.value)

# Faster
```

```

class SomeClass

:
    ...
    def
method(self):
    value = self.value
    for
x in
s:
    op(value)

```

□□□□□□□□

□□□□□□□□□□□□□□□□decorator□□□□  
 □property□□□□□□□descriptor□□□□□□□□□□□□  
 □□□□□□□□□□□□□□□□□□□□□□□□

```

class A

:
    def

__init__(self, x, y):
    self.x = x
    self.y = y
    @property
    def

y(self):
    return

self._y
    @y.setter

```

```
def
y(self, value):
    self._y = value
```

□□□□□□□□□□□□□□

```
>>> from timeit import
timeit
>>> a = A(1,2)
>>> timeit('a.x', 'from __main__ import a')
0.07817923510447145
>>> timeit('a.y', 'from __main__ import a')
0.35766440676525235
>>>
```

□□□□□□□property□□y□□□□□□□□x□□□□□  
□□□□□□□4.5□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□y□□□property□□□□□□□□□□□□□□  
property□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□getter/setter□□□□□□□□□□□□□□□□□□□□  
Python□□□

□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□C□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□

```
values = [x for
x in
sequence]
squares = [x*x for
x in
values]
```

□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□ □□□

```
squares = [x*x for
x in
sequence]
```

Python 的 `copy` 模块提供了 `copy` 和 `deepcopy` 两个函数。其中 `copy` 是浅拷贝，而 `deepcopy` 是深拷贝。Python 的 `copy` 模块位于 `copy` 包中。

### 14.14.3 字典

字典的查找复杂度是  $O(n)$ ，而列表的查找复杂度是  $O(n^2)$ 。这是因为字典内部使用了哈希表结构，而列表则是线性结构。

字典的查找速度非常快，几乎是常数时间。这是因为字典内部使用了哈希表结构，而列表则是线性结构。字典的查找速度非常快，几乎是常数时间。

字典的查找速度非常快，几乎是常数时间。这是因为字典内部使用了哈希表结构，而列表则是线性结构。字典的查找速度非常快，几乎是常数时间。

```
a = {
    'name' : 'AAPL',
    'shares' : 100,
    'price' : 534.22
}

b = dict(name='AAPL', shares=100, price=534.22)
```

字典的查找速度非常快，几乎是常数时间。这是因为字典内部使用了哈希表结构，而列表则是线性结构。字典的查找速度非常快，几乎是常数时间。







## 15 C

Python C Python  
C Python  
Python 2  
Python 3

Python C API  
C  
C  
C  
C

C

```
/* sample.c */  
  
_method  
#include <math.h>  
  
/* Compute the greatest common divisor */  
  
int  
gcd(int  
x, int  
y) {  
    int
```

```

g = y;
    while

(x > 0) {
    g = x;
    x = y % x;
    y = g;
}
    return

g;
}

/* Test if (x0,y0) is in the Mandelbrot set or not */

int
in_mandel(double
x0, double
y0, int
n) {
    double
x=0,y=0,xtemp;
    while

(n > 0) {
    xtemp = x*x - y*y + x0;
    y = 2*x*y + y0;
    x = xtemp;
    n -= 1;
    if

(x*x + y*y > 4) return

0;
}
    return

1;

```

```

}

/* Divide two numbers */

int
divide(int
a, int
b, int
*remainder) {
    int
    quot = a / b;
    *remainder = a % b;
    return
    quot;
}

/* Average values in an array */

double
avg(double
*a, int
n) {
    int
    i;
    double
    total = 0.0;
    for
    (i = 0; i < n; i++) {
        total += a[i];
    }
    return

```

```

total / n;
}

/* A C data structure */

typedef struct
Point {
    double

x,y;
} Point;

/* Function involving a C data structure */

double

distance(Point *p1, Point *p2) {
    return

hypot(p1->x - p2->x, p1->y - p2->y);
}

```

□□□□□□□□C□□□□□□□□□□□□□□□□□□□□□□□□  
 □gcd()□is\_mandel()□□divide()□□C□□□□□□□□□□  
 □□□□□□□□□□□□□□□□□□□□□□□□avg()□□□□□C□□□□  
 □□□□□□□□Point□distance()□□□□□C□□□□□

□□□□□□□□□□□□□□□C□□□□□□□□sample.c □□  
 □□□□□□□□sample.h □□□□□□□□□□□□□□□□  
 libsample□□□□□□□□□□□□□□□C□□□□□□□□□□□□□□□□

Python 2.7.10 64-bit  
C 32-bit

## 15.1 ctypes C

### 15.1.1

C DLL  
Python C  
Python

### 15.1.2

C Python ctypes  
ctypes C  
Python  
libsample.so  
libsample.so sample.py

Python

```
# sample.py
```

```
import ctypes
```

```
import os
```

```
# Try to locate the .so file in the same directory as this  
file
```

```
_file = 'libsample.so'  
_path = os.path.join(*(os.path.split(__file__)[-1] +  
(_file,)))  
_mod = ctypes.cdll.LoadLibrary(_path)
```

```
# int gcd(int, int)
```

```
gcd = _mod.gcd  
gcd.argtypes = (ctypes.c_int, ctypes.c_int)  
gcd.restype = ctypes.c_int
```

```
# int in_mandel(double, double, int)
```

```
in_mandel = _mod.in_mandel  
in_mandel.argtypes = (ctypes.c_double, ctypes.c_double,  
ctypes.c_int)  
in_mandel.restype = ctypes.c_int
```

```
# int divide(int, int, int *)
```

```
_divide = _mod.divide  
_divide.argtypes = (ctypes.c_int, ctypes.c_int,  
ctypes.POINTER(ctypes.c_int))  
_divide.restype = ctypes.c_int
```

```
def
```

```
divide(x, y):  
    rem = ctypes.c_int()
```

```

    quot = _divide(x, y, rem)
    return

quot, rem.value

# void avg(double *, int n)

# Define a special type for the 'double *' argument

class DoubleArrayType

:
    def
from_param(self, param):
    typename = type(param).__name__
    if
hasattr(self, 'from_' + typename):
    return
getattr(self, 'from_' + typename)(param)
    elif
isinstance(param, ctypes.Array):
    return
param
    else
:
    raise TypeError
("Can't convert %s" % typename)

    # Cast from array.array objects

    def
from_array(self, param):
    if

```

```

param.typecode != 'd':
    raise TypeError

('must be an array of doubles')
    ptr, _ = param.buffer_info()
    return

ctypes.cast(ptr, ctypes.POINTER(ctypes.c_double))

    # Cast from lists/tuples

def

from_list(self, param):
    val = ((ctypes.c_double)*len(param))(*param)
    return

val

    from_tuple = from_list

    # Cast from a numpy array

def

from_ndarray(self, param):
    return

param.ctypes.data_as(ctypes.POINTER(ctypes.c_double))

DoubleArray = DoubleArrayType()
_avg = _mod.avg
_avg.argtypes = (DoubleArray, ctypes.c_int)
_avg.restype = ctypes.c_double

def

avg(values):
    return

_avg(values, len(values))

# struct Point { }

```



```
(ctypes.Structure):
    _fields_ = [('x', ctypes.c_double),
                 ('y', ctypes.c_double)]

# double distance(Point *, Point *)

distance = _mod.distance
distance.argtypes = (ctypes.POINTER(Point),
                    ctypes.POINTER(Point))
distance.restype = ctypes.c_double
```

☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ C ☐ ☐ ☐ ☐

```
>>> import sample

>>> sample.gcd(35,42)
7
>>> sample.in_mandel(0,0,500)
1
>>> sample.in_mandel(2.0,1.0,500)
0
>>> sample.divide(42,8)
(5, 2)
>>> sample.avg([1,2,3])
2.0
>>> p1 = sample.Point(1,2)
>>> p2 = sample.Point(4,5)
>>> sample.distance(p1,p2)
4.242640687119285
>>>
```

## 15.1.3 加载

ctypes 模块提供了 Python 加载 C 库的接口。 ctypes 模块提供了 C 库的接口， sample.py 文件中， ctypes 模块加载了 libsample.so 库。 Python 提供了 ctypes 模块， ctypes 模块提供了 C 库的接口， sample.py 文件中， ctypes 模块加载了 libsample.so 库。

ctypes 模块提供了 C 库的接口， ctypes 模块提供了 C 库的接口， ctypes.util.find\_library() 函数用于查找库。

```
>>> from ctypes.util import  
find_library  
>>> find_library('m')  
'/usr/lib/libm.dylib'  
>>> find_library('pthread')  
'/usr/lib/libpthread.dylib'  
>>> find_library('sample')  
'/usr/local/lib/libsample.so'  
>>>
```

ctypes 模块提供了 C 库的接口， ctypes 模块提供了 C 库的接口， ctypes 模块提供了 C 库的接口。

ctypes 模块提供了 C 库的接口， ctypes.cdll.LoadLibrary() 函数用于加载库。

```
_mod = ctypes.cdll.LoadLibrary(_path)
```

```
# int in_mandel(double, double, int)
```

```

    ctypes.argtypes=ctypes.c_int, ctypes.c_double, ctypes.c_float, ctypes.c_char_p
    ctypes.restype=ctypes.c_int
    c_double=c_double, c_int=c_int, c_short=c_short, c_float=c_float, c_char_p=c_char_p
    C=C, Python=Python

```

ctypes  
Python  
divide()  
C  
Python

```

>>> divide = _mod.divide
>>> divide.argtypes = (ctypes.c_int, ctypes.c_int,
ctypes.POINTER(ctypes.c_int))
>>> x = 0
>>> divide(10, 3, x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ctypes.ArgumentError: argument 3: <class 'TypeError'>:
expected LP_c_int
instance instead of int
>>>

```

ctypes Python  
 ctypes Python  
 ctypes

```

>>> x = ctypes.c_int()
>>> divide(10, 3, x)
3
>>> x.value
1
>>>

```

ctypes.c\_int Python  
 Python c\_int  
 .value

C calling convention  
 Pythonic Pythonic Python  
 Python  
 divide()

```
# int divide(int, int, int *)

_divide = _mod.divide
_divide.argtypes = (ctypes.c_int, ctypes.c_int,
ctypes.POINTER(ctypes.c_int))
_divide.restype = ctypes.c_int

def
divide(x, y):
    rem = ctypes.c_int()
    quot = _divide(x,y,rem)
    return
quot, rem.value
```

avg()은 C에서 Python으로  
array, numpy  
Python “”

DoubleArrayType  
from\_param()  
ctypes  
ctypes.c\_double  
from\_param()  
list  
from\_list()

```
from_list() ctypes
ctypes

```

```
>>> nums = [1, 2, 3]
>>> a = (ctypes.c_double * len(nums))(*nums)
>>> a
<__main__.c_double_Array_3 object at 0x10069cd40>
>>> a[0]
1.0
>>> a[1]
2.0
>>> a[2]
3.0
>>>
```

```
array from_array()
ctypes

```

```
>>> import array

>>> a = array.array('d', [1, 2, 3])
>>> a
array('d', [1.0, 2.0, 3.0])
>>> ptr_ = a.buffer_info()
>>> ptr
4298687200
>>> ctypes.cast(ptr, ctypes.POINTER(ctypes.c_double))
<__main__.LP_c_double object at 0x10069cd40>
>>>
```

```
from_ndarray() numpy

```

DoubleArrayType avg()

```
>>> import sample

>>> sample.avg([1,2,3])
2.0
>>> sample.avg((1,2,3))
2.0
>>> import array

>>> sample.avg(array.array('d',[1,2,3]))
2.0
>>> import numpy

>>> sample.avg(numpy.array([1.0,2.0,3.0]))
2.0
>>>
```

C

```
class Point
(ctypes.Structure):
    _fields_ = [('x', ctypes.c_double),
                ('y', ctypes.c_double)]
```





Python CFFI Python CFFI

## 15.2 C

### 15.2.1

Python API C

### 15.2.2

C C

```
/* sample.h */

#include <math.h>

extern int
gcd(int
, int
);
extern int
in_mandel(double
```



```

/* int gcd(int, int) */

static
PyObject *py_gcd(PyObject *self, PyObject *args) {
    int
    x, y, result;

    if
    (!PyArg_ParseTuple(args, "ii", &x, &y)) {
        return
        NULL;
    }
    result = gcd(x,y);
    return
    Py_BuildValue("i", result);
}

/* int in_mandel(double, double, int) */

static
PyObject *py_in_mandel(PyObject *self, PyObject *args) {
    double
    x0, y0;
    int
    n;
    result;

    if
    (!PyArg_ParseTuple(args, "ddi", &x0, &y0, &n)) {
        return
        NULL;
    }

```

```

    }
    result = in_mandel(x0,y0,n);
    return

Py_BuildValue("i", result);
}

/* int divide(int, int, int *) */

static

PyObject *py_divide(PyObject *self, PyObject *args) {
    int

a, b, quotient, remainder;
    if

(!PyArg_ParseTuple(args, "ii", &a, &b)) {
        return

NULL;
    }
    quotient = divide(a,b, &remainder);
    return

Py_BuildValue("(ii)", quotient, remainder);
}

/* Module method table */

static

PyMethodDef SampleMethods[] = {
    {"gcd", py_gcd, METH_VARARGS, "Greatest common divisor"},
    {"in_mandel", py_in_mandel, METH_VARARGS, "Mandelbrot
test"},
    {"divide", py_divide, METH_VARARGS, "Integer division"},
    { NULL, NULL, 0, NULL}
};

/* Module structure */

```

```

static struct

PyModuleDef samplemodule = {
    PyModuleDef_HEAD_INIT,
    "sample",          /* name of module */

    "A sample module", /* Doc string (may be NULL) */

    -1,                /* Size of per-interpreter state or -1 */

    SampleMethods      /* Method table */
};

/* Module initialization function */

PyMODINIT_FUNC
PyInit_sample(void

) {
    return

PyModule_Create(&samplemodule);
}

```

□□□□□□□□□□□□□□□□ *setup.py* □□□□□□□□□□  
 □□

```

# setup.py

from distutils.core import

```

```

setup, Extension

setup(name='sample',
      ext_modules=[
          Extension('sample',
                    ['pysample.c'],
                    include_dirs = ['/some/dir'],
                    define_macros = [('F00','1')],
                    undef_macros = ['BAR'],
                    library_dirs = ['/usr/local/lib'],
                    libraries = ['sample']
                    )
      ]
)

```

python3 buildlib.py  
 build\_ext --inplace

```

bash % python3 setup.py build_ext --inplace
running build_ext
building 'sample' extension
gcc -fno-strict-aliasing -DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes
-I/usr/local/include/python3.3m -c pysample.c
-o build/temp.macosx-10.6-x86_64-3.3/pysample.o
gcc -bundle -undefined dynamic_lookup
build/temp.macosx-10.6-x86_64-3.3/pysample.o \

-L/usr/local/lib -lsample -o sample.so
bash %

```

sample.so  
 Python

```
>>> import sample

>>> sample.gcd(35, 42)
7
>>> sample.in_mandel(0, 0, 500)
1
>>> sample.in_mandel(2.0, 1.0, 500)
0
>>> sample.divide(42, 8)
(5, 2)
>>>
```

Windows Python Visual Studio Python  
<http://docs.python.org/3/extending/windows.html>

### 15.2.3

Python “Python Extending and Embedding the Python Interpreter Python C API API

```
static
```

```
PyObject *py_func(PyObject *self, PyObject *args) {  
    ...  
}
```

PyObject C Python  
C Python  
PyObject \*args Python  
self C  
self

PyArg\_ParseTuple() Python  
C  
"i" "d" double C  
PyArg\_ParseTuple()  
NULL NULL

Py\_BuildValue() C  
Python  
Python Py\_BuildValue()  
Py\_BuildValue()





## 15.3 数组和缓冲区

### 15.3.1 数组

Python 的 `array` 模块和 `NumPy` 模块提供了对 C 语言数组的支持。它们提供了与 C 语言数组类似的操作，但使用 Python 的语法和接口。

### 15.3.2 缓冲区

Python 的 `Buffer Protocol` (<http://docs.python.org/3/c-api/buffer.html>) 提供了一种在 C 语言中操作 Python 对象缓冲区的方法。它允许 C 语言代码通过调用 `avg(double *buf, int len)` 来计算一个缓冲区中元素的平均值。

```
/* Call double avg(double *, int) */

static
PyObject *py_avg(PyObject *self, PyObject *args) {
    PyObject *bufobj;
    Py_buffer view;
    double
    result;
    /* Get the passed Python object */
```

```

    if
(!PyArg_ParseTuple(args, "0", &bufobj)) {
    return
NULL;
}

/* Attempt to extract buffer information from it */

    if
(PyObject_GetBuffer(bufobj, &view,
    PyBUF_ANY_CONTIGUOUS | PyBUF_FORMAT) == -1) {
    return
NULL;
}
    if
(view.ndim != 1) {
    PyErr_SetString(PyExc_TypeError, "Expected a 1-dimensional
array");
    PyBuffer_Release(&view);
    return
NULL;
}

/* Check the type of items in the array */

    if
(strcmp(view.format, "d") != 0) {
    PyErr_SetString(PyExc_TypeError, "Expected an array of
doubles");
    PyBuffer_Release(&view);
    return
NULL;
}

```

```

    /* Pass the raw buffer and size to the C function */

    result = avg(view.buf, view.shape[0]);

    /* Indicate we're done working with the buffer */

    PyBuffer_Release(&view);
    return

Py_BuildValue("d", result);
}

```

□□□□□□□□□□□□□□□□□□□□

```

>>> import array

>>> avg(array.array('d',[1,2,3]))
2.0
>>> import numpy

>>> avg(numpy.array([1.0,2.0,3.0]))
2.0
>>> avg([1,2,3])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'list' does not support the buffer interface
>>> avg(b'Hello')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Expected an array of doubles
>>> a = numpy.array([[1.,2.,3.],[4.,5.,6.]])
>>> avg(a[:,2])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: ndarray is not contiguous
>>> sample.avg(a)
Traceback (most recent call last):

```

```
File "<stdin>", line 1, in <module>
TypeError: Expected a 1-dimensional array
>>> sample.avg(a[0])
2.0
>>>
```

### 15.3.3 ☐ ☐

A diagram consisting of a single horizontal row of 25 square boxes. The 10th box from the left contains the word "Python" in a black, sans-serif font. All other boxes are empty.

```
PyBuffer_GetBuffer() Python
Python——Python
-1 PyBuffer_GetBuffer()
PyBUF_ANY_CONTIGUOUS
```

```
Py_buffer
```

**typedef struct**

```

bufferinfo {
    void

    *buf;                /* Pointer to buffer memory */

    PyObject *obj;        /* Python object that is the
owner */

    Py_ssize_t len;        /* Total size in bytes */

    Py_ssize_t itemsize;    /* Size in bytes of a single
item */

    int

    readonly;            /* Read-only access flag */

    int

    ndim;                /* Number of dimensions */

    char

    *format;            /* struct code of a single item */

    Py_ssize_t *shape;    /* Array containing dimensions
*/

    Py_ssize_t *strides;    /* Array containing strides
*/

    Py_ssize_t *suboffsets; /* Array containing suboffsets
*/
} Py_buffer;

```

double format struct format struct  
double format struct  
"d" struct format struct  
C  
format struct  
C  
format struct

C  
avg() C  
array numpy

PyBuffer\_Release()

<http://docs.python.org/3/c-api/buffer.html>

C  
Cython 15.11

## 15.4 C 라이브러리 호출하기

### 15.4.1 개요

이 장에서는 C 라이브러리를 Python에서 호출하는 방법을 소개한다. C 라이브러리를 호출하는 방법은 Python의 C API를 사용하여 가능하다.

### 15.4.2 예제

이 예제는 C 라이브러리를 호출하여 두 점 사이의 거리를 계산하는 capsule을 보여준다. 이 예제는 C 라이브러리를 호출하여 두 점 사이의 거리를 계산하는 capsule을 보여준다.

```
typedef struct
Point {
    double
x,y;
} Point;

extern double
distance(Point *p1, Point *p2);
```

이 예제는 C 라이브러리를 호출하여 두 점 사이의 거리를 계산하는 capsule을 보여준다. 이 예제는 C 라이브러리를 호출하여 두 점 사이의 거리를 계산하는 capsule을 보여준다.

```
/* Destructor function for points */
```



```

static void
del_Point(PyObject *obj) {
    free(PyCapsule_GetPointer(obj,"Point"));
}

/* Utility functions */

static
Point *PyPoint_AsPoint(PyObject *obj) {
    return

    (Point *) PyCapsule_GetPointer(obj, "Point");
}

static
PyObject *PyPoint_FromPoint(Point *p, int
must_free) {
    return

    PyCapsule_New(p, "Point", must_free ? del_Point : NULL);
}

/* Create a new Point object */

static
PyObject *py_Point(PyObject *self, PyObject *args) {
    Point *p;
    double

    x,y;
    if

    (!PyArg_ParseTuple(args,"dd",&x,&y)) {
        return

        NULL;
    }
}

```

```

    }
    p = (Point *) malloc(sizeof
(Point));
    p->x = x;
    p->y = y;
    return
PyPoint_FromPoint(p, 1);
}

static
PyObject *py_distance(PyObject *self, PyObject *args) {
    Point *p1, *p2;
    PyObject *py_p1, *py_p2;
    double
result;

    if
(!PyArg_ParseTuple(args, "00",&py_p1, &py_p2)) {
        return
NULL;
    }
    if
(!((p1 = PyPoint_AsPoint(py_p1))) {
        return
NULL;
    }
    if
(!((p2 = PyPoint_AsPoint(py_p2))) {
        return
NULL;
    }
    result = distance(p1,p2);
    return
Py_BuildValue("d", result);

```

```
}
```

## Python

```
>>> import sample

>>> p1 = sample.Point(2,3)
>>> p2 = sample.Point(4,5)
>>> p1
<capsule object "Point" at 0x1004ea330>
>>> p2
<capsule object "Point" at 0x1005d1db0>
>>> sample.distance(p1,p2)
2.8284271247461903
>>>
```

### 15.4.3

capsule C void capsule  
PyCapsule\_New() capsule  
capsule capsule capsule  
capsule capsule capsule

capsule  
PyCapsule\_GetPointer() capsule  
capsule capsule  
NULL

PyPoint\_FromPoint()
 PyPoint\_AsPoint()
 capsule
 Point
 capsule
 Point
 capsule

capsule
 PyPoint\_FromPoint()
 must\_free
 capsule
 Point
 C
 ownership
 Point
 capsule
 PyCapsule\_SetDestructor()

C
 capsules
 capsule

## 15.5 C API

### 15.5.1



```
PyCapsule_New(p, "Point", must_free ? del_Point : NULL);
}
```

PyPoint\_AsPoint() API  
PyPoint\_FromPoint() API  
Point

*pysample.h*

```
/* pysample.h */

#include "Python.h"
#include "sample.h"
#ifdef __cplusplus
extern
"C" {
#endif

/* Public API Table */

typedef struct
{
    Point *(*aspoint)(PyObject *);
    PyObject *(*frompoint)(Point *, int);
} _PointAPIMethods;

#ifdef PYSAMPLE_MODULE
/* Method table in external module */
```

```

static
_PointAPIMethods *_point_api = 0;

/* Import the API table from sample */

static int
import_sample(void
) {
    _point_api = (_PointAPIMethods *)
PyCapsule_Import("sample._point_api",0);
    return

(_point_api != NULL) ? 1 : 0;
}

/* Macros to implement the programming interface */

#define PyPoint_AsPoint(obj) (_point_api->aspoint)(obj)
#define PyPoint_FromPoint(obj) (_point_api->frompoint)(obj)
#define

#ifdef __cplusplus
}
#endif

```

□□□□□□□□□□□□□□□□\_PointAPIMethods□□  
 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```

/* pysample.c */

```

```

#include "Python.h"
#define PYSAMPLE_MODULE
#include "pysample.h"

...
/* Destructor function for points */

static void
del_Point(PyObject *obj) {
    printf("Deleting point\n
");
    free(PyCapsule_GetPointer(obj, "Point"));
}

/* Utility functions */

static
Point *PyPoint_AsPoint(PyObject *obj) {
    return
(Point *) PyCapsule_GetPointer(obj, "Point");
}

static
PyObject *PyPoint_FromPoint(Point *p, int
free) {
    return
PyCapsule_New(p, "Point", free ? del_Point : NULL);
}

static _PointAPIMethods _point_api = {
    PyPoint_AsPoint,
    PyPoint_FromPoint
};

```





```

(!PyArg_ParseTuple(args, "O", &obj)) {
    return
    NULL;
}

/* Note: This is defined in a different module */

p = PyPoint_AsPoint(obj);
if
(!p) {
    return
    NULL;
}
printf("%f %f\n", p->x, p->y);
return
Py_BuildValue("");
}

static

PyMethodDef PtExampleMethods[] = {
    {"print_point", print_point, METH_VARARGS, "output a
point"},
    { NULL, NULL, 0, NULL}
};

static struct

PyModuleDef ptexamplemodule = {
    PyModuleDef_HEAD_INIT,
    "ptexample",
    /* name of module */

    "A module that imports an API",
    /* Doc string (may be
NULL) */

```

```

    -1,                                /* Size of per-interpreter
state or -1 */

    PtExampleMethods                    /* Method table */

};

/* Module initialization function */

PyMODINIT_FUNC
PyInit_ptexample(void

) {
    PyObject *m;

    m = PyModule_Create(&ptexamplemodule);
    if

(m == NULL)
    return

NULL;

    /* Import sample, loading its API functions */

    if

(!import_sample()) {
        return

NULL;
    }

    return

m;
}

```

`setup.py`

```
# setup.py

from distutils.core import
setup, Extension

setup(name='ptexample',
      ext_modules=[
          Extension('ptexample',
                  ['ptexample.c'],
                  include_dirs = [], # May need pysample.h
                  directory
                  )
      ]
)
```

**C API**

```
>>> import sample

>>> p1 = sample.Point(2,3)
>>> p1
<capsule object "Point *" at 0x1004ea330>
>>> import ptextample

>>> ptextample.print_point(p1)
2.000000 3.000000
>>>
```

### 15.5.3

```
capsule
capsule
capsule
capsule
sample._point_api
```

```
Python
PyCapsule_Import()
sample._point_api
capsule
```

```
C
pysample.h _point_api
import_sample()
capsule _point_api
import_sample()
C
API
```



# Python

```
#include <Python.h>

/* Execute func(x,y) in the Python interpreter. The
   arguments and return result of the function must
   be Python floats */

double
call_func(PyObject *func, double
x, double
y) {
    PyObject *args;
    PyObject *kwargs;
    PyObject *result = 0;
    double
retval;

    /* Make sure we own the GIL */

    PyGILState_STATE state = PyGILState_Ensure();

    /* Verify that func is a proper callable */

    if
(!PyCallable_Check(func)) {
        fprintf(stderr, "call_func: expected a callable\n
");
        goto

```

```

fail;
}
/* Build arguments */

args = Py_BuildValue("(dd)", x, y);
kwargs = NULL;

/* Call the function */

result = PyObject_Call(func, args, kwargs);
Py_DECREF(args);
Py_XDECREF(kwargs);

/* Check for Python exceptions (if any) */

if
(PyErr_Occurred()) {
    PyErr_Print();
    goto
fail;
}

/* Verify the result is a float object */

if
(!PyFloat_Check(result)) {
    fprintf(stderr, "call_func: callable didn't return a
float\n
");
    goto
fail;
}

/* Create the return value */

```



```

    retval = PyFloat_AsDouble(result);
    Py_DECREF(result);

    /* Restore previous GIL state and return */

    PyGILState_Release(state);
    return

retval;

fail:
    Py_XDECREF(result);
    PyGILState_Release(state);
    abort(); // Change to something more appropriate
}

```

Python

Python

```

#include <Python.h>

/* Definition of call_func() same as above */

...
/* Load a symbol from a module */

PyObject *import_name(const char

```

```

*modname, const char

*symbol) {
    PyObject *u_name, *module;
    u_name = PyUnicode_FromString(modname);
    module = PyImport_Import(u_name);
    Py_DECREF(u_name);
    return

PyObject_GetAttrString(module, symbol);
}

/* Simple embedding example */

int

main() {
    PyObject *pow_func;
    double

x;

    Py_Initialize();
    /* Get a reference to the math.pow function */

    pow_func = import_name("math","pow");

    /* Call it using our call_func() code */

    for

(x = 0.0; x < 10.0; x += 0.1) {
    printf("%0.2f %0.2f\n"

", x, call_func(pow_func,x,2.0));
    }
    /* Done */

    Py_DECREF(pow_func);
    Py_Finalize();

```

```
    return  
  
    0;  
}
```

编译选项和Python解释器  
Makefile中指定编译选项和Python解释器  
编译选项

```
all::  
    cc -g embed.c -I/usr/local/include/python3.3m \  
        -L/usr/local/lib/python3.3/config-3.3m -lpthon3.3m
```

编译选项和Python解释器

```
0.00 0.00  
0.10 0.01  
0.20 0.04  
0.30 0.09  
0.40 0.16  
...
```

编译选项和Python解释器  
call\_func()函数

```
/* Extension function for testing the C-Python callback */  
PyObject *py_call_func(PyObject *self, PyObject *args) {  
    PyObject *func;  
    double x, y, result;
```

```

    if (!PyArg_ParseTuple(args, "Odd", &func, &x, &y)) {
        return NULL;
    }
    result = call_func(func, x, y);
    return Py_BuildValue("d", result);
}

```

[illegible]

```
>>> import sample
>>> def add(x,y):
...     return x+y
...
>>> sample.call_func(add,3,4)
7.0
>>>
```

### 15.6.3 ☐ ☐

**CPython**

```

Python
__call__()
PyCallable Check()

```

```
double call_func(PyObject *func, double x, double y) {
    ...
    /* Verify that func is a proper callable */
    if (!PyCallable_Check(func)) {
```

```

    fprintf(stderr, "call_func: expected a callable\n");
    goto fail;
}
...

```

CPythonのC実装は、PythonのC実装と異なり、gotoとabort()を使用している。これは、Cの実装が、Pythonの実装よりも、より低レベルな操作を行うためである。

PyObject\_Call()は、PythonのC実装で、PyObject\_Call()という関数を実装している。Py\_BuildValue()は、PythonのC実装で、Py\_BuildValue()という関数を実装している。

```

double call_func(PyObject *func, double x, double y) {
    PyObject *args;
    PyObject *kwargs;

    ...
    /* Build arguments */
    args = Py_BuildValue("(dd)", x, y);
    kwargs = NULL;

    /* Call the function */
    result = PyObject_Call(func, args, kwargs);
    Py_DECREF(args);
    Py_XDECREF(kwargs);
    ...
}

```

NULL  
Py\_DECREF() Py\_XDECREF()  
NULL

Python  
PyErr\_Occurred()  
C Python  
abort() C

```
...
/* Check for Python exceptions (if any) */
if (PyErr_Occurred()) {
    PyErr_Print();
    goto fail;
}
...
fail:
    PyGILState_Release(state);
    abort();
```

Python  
Python  
concrete <https://docs.python.org/3/c-api/concrete.html>

PyFloat\_Check() Py\_Float\_AsDouble() Python

C Python Python GIL C Python GIL Python PyGILState\_Ensure() PyGILState\_Release()

```
double call_func(PyObject *func, double x, double y) {
    ...
    double retval;

    /* Make sure we own the GIL */
    PyGILState_STATE state = PyGILState_Ensure();
    ...
    /* Code that uses Python C API functions */
    ...
    /* Restore previous GIL state and return */
    PyGILState_Release(state);
    return retval;
fail:
    PyGILState_Release(state);
    abort();
}
```

PyGILState\_Ensure() Python C Python C Python C-API PyGILState\_Release()

PyGILState\_Ensure() PyGILState\_Release() goto fail: Python finally:

C GIL Python C Python

## 15.7 C GIL

### 15.7.1

C Python GIL

### 15.7.2

C GIL

```
#include "Python.h"
...
Py0bject *pyfunc(Py0bject *self, Py0bject *args) {
    ...
}
```



```

Py_BEGIN_ALLOW_THREADS
// Threaded C code.  Must not use Python API functions
...
Py_END_ALLOW_THREADS
...
return result;
}

```

### 15.7.3 `Py_BEGIN_ALLOW_THREADS`

`Py_BEGIN_ALLOW_THREADS` and `Py_END_ALLOW_THREADS` are used to temporarily disable the Global Interpreter Lock (GIL) in Python C API code. This is useful for C code that needs to perform long-running operations without being interrupted by the Python interpreter. For example, when using NumPy for numerical computations or performing I/O operations, disabling the GIL can significantly improve performance.

The GIL is a Python interpreter lock that allows only one thread to execute Python code at a time. When the GIL is disabled, multiple threads can execute Python code simultaneously. This is achieved by calling `Py_BEGIN_ALLOW_THREADS` before the C code and `Py_END_ALLOW_THREADS` after it. The GIL is then re-enabled, and the Python interpreter continues its execution.

## 15.8 `Py_BEGIN_ALLOW_THREADS` and `Py_END_ALLOW_THREADS`

### 15.8.1 `Py_BEGIN_ALLOW_THREADS`

The `Py_BEGIN_ALLOW_THREADS` macro is used to temporarily disable the GIL in Python C API code. It is typically used in conjunction with `Py_END_ALLOW_THREADS` to ensure that the GIL is re-enabled after the C code has finished executing. This is useful for C code that needs to perform long-running operations without being interrupted by the Python interpreter. For example, when using NumPy for numerical computations or performing I/O operations, disabling the GIL can significantly improve performance.

## 15.8.2 线程

在Python C API中，Python解释器在启动时会初始化线程，并会释放GIL。在Python C API中，Python解释器在启动时会初始化线程，并会释放GIL。在Python C API中，Python解释器在启动时会初始化线程，并会释放GIL。

```
#include <Python.h>

...
if
(!PyEval_ThreadsInitialized()) {
    PyEval_InitThreads();
}
...
```

在Python C API中，Python解释器在启动时会初始化线程，并会释放GIL。在Python C API中，Python解释器在启动时会初始化线程，并会释放GIL。在Python C API中，Python解释器在启动时会初始化线程，并会释放GIL。

```
...
/* Make sure we own the GIL */

PyGILState_STATE state = PyGILState_Ensure();

/* Use functions in the interpreter */

...
/* Restore previous GIL state and return */
```

```
PyGILState_Release(state);
...
```

```
PyGILState_Ensure()
PyGILState_Release()
```

### 15.8.3

`CPython`

—`CPython`

`Python`

`GIL`

```

    PyGILState_Ensure()
    # ...
    GIL
    Python
    # ...

```

## 15.9 Swig C

## 15.9.1 ☐☐


<http://www.swig.org>

## 15.9.2 例題

Swigを使ってCライブラリをPythonから呼び出す例を示す。

```
/* sample.h */

#include <math.h>
extern int
gcd(int
, int
);
extern int
in_mandel(double
x0, double
y0, int
n);
extern int
divide(int
a, int
b, int
*remainder);
extern double
avg(double
*a, int
```

```

n);

typedef struct
Point {
    double

x,y;
} Point;

extern double

distance(Point *p1, Point *p2);

```

1. 编译并安装 Swig“”
 2. 编译并安装

```

// sample.i - Swig interface

%module sample
%{
#include "sample.h"
%}

/* Customizations */

%extend Point {
    /* Constructor for Point objects */

    Point(double

x, double

y) {
        Point *p = (Point *) malloc(sizeof

```

```

(Point));
    p->x = x;
    p->y = y;
    return

p;
    };
};

/* Map int *remainder as an output argument */

#include typemaps.i
%apply int

*OUTPUT { int

* remainder };

/* Map the argument pattern (double *a, int n) to arrays */

%typemap(in) (double

*a, int

n)(Py_buffer view) {
    view.obj = NULL;
    if

(PyObject_GetBuffer($input, &view, PyBUF_ANY_CONTIGUOUS |
PyBUF_FORMAT) == -1){
        SWIG_fail;
    }
    if

(strcmp(view.format,"d") != 0) {
        PyErr_SetString(PyExc_TypeError, "Expected an array of
doubles");
        SWIG_fail;
    }
    $1 = (double

*) view.buf;

```

```

    $2 = view.len / sizeof
(double
);
}

%typemap(freearg) (double
*a, int
n) {
    if
(view$argnum.obj) {
    PyBuffer_Release(&view$argnum);
    }
}

/* C declarations to be included in the extension module */

extern int
gcd(int
, int
);
extern int
in_mandel(double
x0, double
y0, int
n);
extern int
divide(int
a, int

```

```

b, int
*remainder);
extern double

avg(double
*a, int
n);

typedef struct
Point {
    double

x,y;
} Point;

extern double

distance(Point *p1, Point *p2);

```

1. 编译并运行 Swig 生成的 Python 接口

```

bash % swig -python -py3 sample.i
bash %

```

2. 生成 C 语言接口 *sample\_wrap.c* 并编译成 *sample.py*

3. 生成 C 语言接口 *\_sample* 并编译成 *setup.py*



```
# setup.py

from distutils.core import
setup, Extension

setup(name='sample',
      py_modules=['sample.py'],
      ext_modules=[
          Extension('_sample',
                  ['sample_wrap.c'],
                  include_dirs = [],
                  define_macros = [],
                  undef_macros = [],
                  library_dirs = [],
                  libraries = ['sample']
                  )
      ]
)
```

1. 在终端中运行 `python3 setup.py build_ext --inplace`

```
bash % python3 setup.py build_ext --inplace
running build_ext
building '_sample' extension
gcc -fno-strict-aliasing -DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes
-I/usr/local/include/python3.3m -c sample_wrap.c
-o build/temp.macosx-10.6-x86_64-3.3/sample_wrap.o
sample_wrap.c: In function
'SWIG_InitializeModule':
sample_wrap.c:3589: warning: statement with no effect
gcc -bundle -undefined dynamic_lookup build/temp.macosx-10.6-x86_64-3.3/sample.o
build/temp.macosx-10.6-x86_64-3.3/sample_wrap.o -o _sample.so
-lsample
```

bash %

☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☒ C ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

☐ ☐

```
>>> import sample

>>> sample.gcd(42,8)
2
>>> sample.divide(42,8)
[5, 2]
>>> p1 = sample.Point(2,3)
>>> p2 = sample.Point(4,5)
>>> sample.distance(p1,p2)
2.8284271247461903
>>> p1.x
2.0
>>> p1.y
3.0
>>> import array

>>> a = array.array('d',[1,2,3])
>>> sample.avg(a)
2.0
>>>
```

### 15.9.3

Swig Python 1.4 Python 3 Swig Python

C  
 Python  
 Swig

# Swig

```
%module sample
%{
#include "sample.h"
%}
```

[illegible]

Swig C

```
%module sample
%{
#include "sample.h"
%}

...
extern int

gcd(int

, int

);
```



```
SwigC

```

```
>>> p1 = sample.Point(2,3)
>>>
```

```
>>> # Usage if %extend Point is omitted

>>> p1 = sample.Point()
>>> p1.x = 2.0
>>> p1.y = 3
```

```

    typemaps.i%apply
    %applySwigint remainder
    //////////////////////////////////////
    int remainder////////////////////////////////
    divide()

```

```
>>> sample.divide(42,8)
[5, 2]
>>>
```

%%typemap  
%typemap  
%typemap  
(double \*a, int n)%typemap  
C%Swig%Python%  
%Python%buffer%  
%double%NumPy%  
array%15.3%

%typemap\$1\$2  
%typemapC  
\$1double *a* \$2*int n* \$input  
PyObject %argument  
%

%typemap%Swig  
Python C API  
Swig%Swig  
%

C%Python%  
Swig%Swig  
C%

Swig  
<http://www.swig.org> Python  
<http://www.swig.org/Doc2.0/Python.html>

## 15.10 Cython C

## 15.10.1 ☐ ☐

Python  
Cython  
C

## 15.10.2 ☐☐☐☐

Cython

C — Python

```

C
libsample
csample.pxd

```

```
# csample.pxd
```

```
#

# Declarations of "external" C functions and structures

cdef extern from "sample.h":
    int gcd(int, int)
    bint in_mandel(double, double, int)
    int divide(int, int, int *)
    double avg(double *, int) nogil

    ctypedef struct Point:
        double x
        double y

    double distance(Point *, Point *)
```

Cython C
   
 cdef extern from "sample.h"
   
 C
   
*csample.pxd* *sample.pxd* —

*sample.pyx*
  
 Python *csample.pxd*
  
 C

```
# sample.pyx

# Import the low-level C declarations
```



```

cimport csample

# Import some functionality from Python and the C stdlib

from cpython.pycapsule cimport

*
from libc.stdlib cimport malloc
, free

# Wrappers

def
gcd(unsigned int x, unsigned int y):
    return
csample.gcd(x, y)

def
in_mandel(x, y, unsigned int n):
    return
csample.in_mandel(x, y, n)

def
divide(x, y):
    cdef int rem
    quot = csample.divide(x, y, &rem)
    return
quot, rem

def
avg(double[:] a):
    cdef:
        int sz

```

```

        double result

sz = a.size
with

nogil:
    result = csample.avg(<double *> &a[0], sz)
return

result

# Destructor for cleaning up Point objects

cdef del_Point(object obj):
    pt = <csample.Point *> PyCapsule_GetPointer(obj,"Point")
    free(<void *> pt)

# Create a Point object and return as a capsule

def

Point(double x,double y):
    cdef csample.Point *p
    p = <csample.Point *> malloc(sizeof(csample.Point))
    if

p == NULL:
    raise MemoryError

("No memory to make a Point")
    p.x = x
    p.y = y
    return

PyCapsule_New(<void *>p,"Point",
<PyCapsule_Destructor>del_Point)

def

distance(p1, p2):
    pt1 = <csample.Point *> PyCapsule_GetPointer(p1,"Point")
    pt2 = <csample.Point *> PyCapsule_GetPointer(p2,"Point")
    return

```

```
csample.distance(pt1,pt2)
```

`setup.py`

```
from distutils.core import
setup
from distutils.extension import
Extension
from Cython.Distutils import
build_ext

ext_modules = [
    Extension('sample',
              ['sample.pyx'],
              libraries=['sample'],
              library_dirs=['.'])]

setup(
    name = 'Sample extension module',
    cmdclass = {'build_ext': build_ext},
    ext_modules = ext_modules
)
```

[illegible]

```
bash % python3 setup.py build_ext --inplace
running build_ext
cythoning sample.pyx to sample.c
building 'sample' extension
gcc -fno-strict-aliasing -DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes
```

```
-I/usr/local/include/python3.3m -c sample.c
-o build/temp.macosx-10.6-x86_64-3.3/sample.o
gcc -bundle -undefined dynamic_lookup build/temp.macosx-10.6-x86_64-3.3/sample.o
-L. -lsample -o sample.so
bash %
```

sample.so

```
>>> import sample

>>> sample.gcd(42,10)
2
>>> sample.in_mandel(1,1,400)
False
>>> sample.in_mandel(0,0,400)
True
>>> sample.divide(42,10)
(4, 2)
>>> import array

>>> a = array.array('d',[1,2,3])
>>> sample.avg(a)
2.0
>>> p1 = sample.Point(2,3)
>>> p2 = sample.Point(4,5)
>>> p1
<capsule object "Point" at 0x1005d1e70>
>>> p2
<capsule object "Point" at 0x1005d1ea0>
>>> sample.distance(p1,p2)
2.8284271247461903
>>>
```

## 15.10.3 并行

并行编程的一个重要问题是互斥。在并行编程中，多个线程同时访问共享资源，可能会导致数据不一致。Python 的 GIL 限制了并行编程的能力，因为它不允许同时执行多个 Python 解释器实例。

Cython 是一个将 Python 代码编译成 C 代码的工具。它允许你编写 C 代码，然后使用 Cython 编译器将其编译成 C 代码。Cython 的语法与 Python 非常相似，但使用 C 的语法。Cython 的语法与 Python 非常相似，但使用 C 的语法。Cython 的语法与 Python 非常相似，但使用 C 的语法。

Cython 的语法与 Python 非常相似，但使用 C 的语法。Cython 的语法与 Python 非常相似，但使用 C 的语法。Cython 的语法与 Python 非常相似，但使用 C 的语法。

```
cimport csample

def
gcd(unsigned int x, unsigned int y):
    return
csample.gcd(x,y)
```

Cython 是一个将 Python 代码编译成 C 代码的工具。它允许你编写 C 代码，然后使用 Cython 编译器将其编译成 C 代码。Cython 的语法与 Python 非常相似，但使用 C 的语法。

#####  
#####

```
>>> sample.gcd(-10,2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "sample.pyx", line 7, in sample.gcd (sample.c:1284)
    def

gcd(unsigned int x,unsigned int y):
OverflowError: can't convert negative value to unsigned int
>>>
```

#####  
####

```
def
gcd(unsigned int x, unsigned int y):
    if
x <= 0:
    raise ValueError
("x must be > 0")
    if
y <= 0:
    raise ValueError
("y must be > 0")
    return
csample.gcd(x,y)
```

csample.pxd in\_mandel() bint  
int  
0 False 1 True

Cython Python  
C divide()

```
def
divide(x,y):
    cdef int rem
    quot = csample.divide(x,y,&rem)
    return
quot, rem
```

rem C int  
divide() &rem rem C

avg() Cython  
def avg(double[:] a) avg()  
double memoryview  
avg numpy

```

>>> import array

>>> a = array.array('d',[1,2,3])
>>> import numpy

>>> b = numpy.array([1., 2., 3.])
>>> import sample

>>> sample.avg(a)
2.0
>>> sample.avg(b)
2.0
>>>

```

□□□□□□□□a.size□&a[0]□□□□□□□□□□□□□□□□  
 □□□□□□□□<double \*> &a[0]□□□□□□□□□□□□  
 □□□□□□□□□□□□□□C□□avg()□□□□□□□□□□□□□□□□  
 □□□□□□□□□□□□□□Cython□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□avg()□□□□□□□□□□□□□□□□  
 □GIL□□□□□□□□with nogil:□□□□□□□□□□□□□□□□  
 □GIL□□□□□□□□□□□□□□□□□□Python□□□□□□□□——□  
 □□□□cdef□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
 □□□□GIL□□□□□□□□*csample.pxd* □□□□□□avg()□  
 □□□double avg(double \*, int) nogil□

□□□□Point□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
 capsule□□□Point□□□□□□□□□□□□□□□□□□□□□□□□□□□□



## 15.4 Cython C API

```
from cpython.pycapsule cimport *
from libc.stdlib cimport malloc, free
```

```
def del_Point(Point):
    capsule = Point * 1
    del_Point()
    Cython
    Python
    capsule
    PyCapsule_New()
    PyCapsule_GetPointer()
    Python C API
```

```
distance(Point):
    capsule =
    PyCapsule_GetPointer()
    Cython
    distance()
```

```
Point

```

```
# sample.pyx
```

```

cimport csample
from libc.stdlib cimport malloc
, free

...

cdef class Point
:
    cdef csample.Point *_c_point
    def
__cinit__(self, double x, double y):
    self._c_point = <csample.Point *>
malloc(sizeof(csample.Point))
    self._c_point.x = x
    self._c_point.y = y

    def
__dealloc__(self):
    free(self._c_point)

    property x:
        def
__get__(self):
            return
self._c_point.x
        def
__set__(self, value):
            self._c_point.x = value

    property y:
        def
__get__(self):
            return
self._c_point.y

```

```

    def
__set__(self, value):
    self._c_point.y = value

def
distance(Point p1, Point p2):
    return
csample.distance(p1._c_point, p2._c_point)

```

```

cdef class Point:
    Point()
    cdef csample.Point * _c_point
    Point() __cinit__() __dealloc__()
    malloc() free()
    property x
    property y
    get set distance()
    Point()

```

```

Point()

```


```

>>> import sample

>>> p1 = sample.Point(2,3)
>>> p2 = sample.Point(4,5)
>>> p1
<sample.Point object at 0x100447288>
>>> p2
<sample.Point object at 0x1004472a0>
>>> p1.x
2.0

```

```
>>> p1.y
3.0
>>> sample.distance(p1,p2)
2.8284271247461903
>>>
```


  
<http://docs.cython.org>

□□□□□□□□□□□□□□□□Cython□□□

## 15.11 Cython

## 15.11.1 ☐ ☐

NumPy Cython

## 15.11.2 □□□□

```

double

```

```
# sample.pyx (Cython)
```

```

cimport cython

@cython.boundscheck(False)
@cython.wraparound(False)
cpdef clip(double[:] a, double min, double max, double[:]
out):
    ...
    Clip the values in a to be between min and max. Result in
out
    ...
    if min > max:
        raise ValueError("min must be <= max")
    if a.shape[0] != out.shape[0]:
        raise ValueError("input and output arrays must be the
same size")
    for i in range(a.shape[0]):
        if a[i] < min:
            out[i] = min
        elif a[i] > max:
            out[i] = max
        else:
            out[i] = a[i]

```

1. Create a `setup.py` file in the same directory as the `sample.pyx` file.

```

python3 setup.py build_ext --inplace

```

2. Run the command `python3 setup.py build_ext --inplace` in the terminal.

```

from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext

ext_modules = [
    Extension('sample',
              ['sample.pyx'])
]

setup(
    name = 'Sample app',
    cmdclass = {'build_ext': build_ext},
    ext_modules = ext_modules
)

```

)

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□

```
>>> # array module example
>>> import sample
>>> import array
>>> a = array.array('d',[1,-3,4,7,2,0])
>>> a

array('d', [1.0, -3.0, 4.0, 7.0, 2.0, 0.0])
>>> sample.clip(a,1,4,a)
>>> a
array('d', [1.0, 1.0, 4.0, 4.0, 2.0, 1.0])

>>> # numpy example
>>> import numpy
>>> b = numpy.random.uniform(-10,10,size=1000000)
>>> b
array([-9.55546017,  7.45599334,  0.69248932, ...,  0.69583148,
        -3.86290931,  2.37266888])
>>> c = numpy.zeros_like(b)
>>> c
array([ 0.,  0.,  0., ...,  0.,  0.,  0.])
>>> sample.clip(b,-5,5,c)
>>> c
array([-5. ,  5. ,  0.69248932, ...,  0.69583148,
        -3.86290931,  2.37266888])
>>> min(c)
-5.0
>>> max(c)
5.0
>>>
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□numpy□□□□□clip()□□□□□□□□□□□□



NumPy numpy.zeros() numpy.zeros\_like() numpy.empty() numpy.empty\_like()

a[i] out[i] Cython

clip() @cython.boundscheck (False) @cython.wraparound(False) 2.5

clip()

```
@cython.boundscheck(False)
@cython.wraparound(False)
cpdef clip(double[:] a, double min, double max, double[:]
out):
    if
min > max:
```





```

n, double
min, double
max, double

*out) {
    double
    x;
    for
    (; n >= 0; n--, a++, out++) {
        x = *a;
        *out = x > max ? max : (x < min ? min : x);
    }
}

```

100% Python code can be converted to C code  
 Cython can convert Python code to C code  
 Cython can convert Python code to C code

100% Python code can be converted to C code  
 GIL (Global Interpreter Lock) is a lock that prevents multiple threads from executing Python code simultaneously  
 with nogil:

```

@cython.boundscheck(False)
@cython.wraparound(False)
cpdef clip(double[:] a, double min, double max, double[:]
out):
    if
    min > max:
        raise ValueError

```



```

range(a.ndim):
    if
a.shape[n] != out.shape[n]:
    raise TypeError

("a and out have different shapes")
    for

i in
range(a.shape[0]):
    for

j in
range(a.shape[1]):
    if

a[i,j] < min:
        out[i,j] = min
    elif

a[i,j] > max:
        out[i,j] = max
    else

:
        out[i,j] = a[i,j]

```

NumPy
 <http://www.python.org/dev/peps/pep-3118>
 Cython
 <http://docs.cython.org/src/>

[userguide/memoryviews.html](http://userguide/memoryviews.html) `memoryviews.html`  
`memoryviews`

## 15.12 `ctypes`

### 15.12.1 `ctypes`

`ctypes` `C` `Python` `ctypes`

### 15.12.2 `ctypes`

`ctypes` `C` `ctypes`

```
>>> import ctypes

>>> lib = ctypes.cdll.LoadLibrary(None)
>>> # Get the address of sin() from the C math library

>>> addr = ctypes.cast(lib.sin, ctypes.c_void_p).value
>>> addr
140735505915760

>>> # Turn the address into a callable function

>>> functype = ctypes.CFUNCTYPE(ctypes.c_double,
ctypes.c_double)
```

```

>>> func = func_type(addr)
>>> func
<CFunctionType object at 0x1006816d0>

>>> # Call the resulting function

>>> func(2)
0.9092974268256817
>>> func(0)
0.0
>>>

```

## 15.12.3 小结

在 LLVM 中，`CFunctionType` 是 `CFunctionType()` 的别名，用于表示 C 函数类型。在 `ctypes` 模块中，`CFUNCTYPE` 是 `CFunctionType` 的别名。

LLVM 提供了 `just in-time compilation` 功能，可以在运行时进行编译。

有关 LLVM 的更多信息，请访问 <http://www.llvmpy.org>。LLVM 是一个开源的编译器基础设施，支持多种编程语言，包括 Python。



## 15.13 用NULL表示空字符串C

### 15.13.1

用NULL表示空字符串C  
PythonUnicode

### 15.13.2

用C表示空字符串char  
\*C

```
void
print_chars(char
*s) {
    while
(*s) {
        printf("%2x ", (unsigned char
) *s);
        s++;
    }
    printf("\n
");
}
```





```
>>> print_chars(b'Hello\x00
World')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be bytes without null bytes, not bytes
>>> print_chars('Hello World')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' does not support the buffer interface
>>>
```

Unicode "s"

```
static
PyObject *py_print_chars(PyObject *self, PyObject *args) {
    char
    *s;

    if
    (!PyArg_ParseTuple(args, "s", &s)) {
        return
        NULL;
    }
    print_chars(s);
    Py_RETURN_NONE;
}
```

NULL

UTF-8

```

>>> print_chars('Hello World')
48 65 6c 6c 6f 20 57 6f 72 6c 64
>>> print_chars('Spicy Jalape\u00f1
o') # Note: UTF-8 encoding

53 70 69 63 79 20 4a 61 6c 61 70 65 c3 b1 6f
>>> print_chars('Hello\x00
World')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str without null characters, not str
>>> print_chars(b'Hello World')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not bytes
>>>

```

```

PyObject *
PyArg_ParseTuple()
char *

```

```

/* Some Python Object (obtained somehow) */

PyObject *obj;

/* Conversion from bytes */

{
    char

*s;
    s = PyBytes_AsString(o);
    if

```



PyArg\_ParseTuple(NULL, &C)

PyArg\_ParseTuple(s, "s", &UTF-8, ASCII)

```
>>> import sys
>>> s = 'Spicy Jalape\u00f1o'
>>> sys.getsizeof(s)
87
>>> print_chars(s) # Passing string
53 70 69 63 79 20 4a 61 6c 61 70 65 c3 b1 6f
>>> sys.getsizeof(s) # Notice increased size
103
>>>
```

PyUnicode\_AsUTF8String(C)

```
static PyObject *py_print_chars(PyObject *self, PyObject
*args) {
    PyObject *o, *bytes;
    char *s;

    if (!PyArg_ParseTuple(args, "U", &o)) {
```

```

        return NULL;
    }
    bytes = PyUnicode_AsUTF8String(o);
    s = PyBytes_AsString(bytes);
    print_chars(s);
    Py_DECREF(bytes);
    Py_RETURN_NONE;
}

```

UTF-8

```
>>> import sys
>>> s = 'Spicy Jalape\u00f1o'
>>> sys.getsizeof(s)
87
>>> print_chars(s)
53 70 69 63 79 20 4a 61 6c 61 70 65 c3 b1 6f
>>> sys.getsizeof(s)
87
>>>
```

```

    NULL ctypes
    ctypes
    NULL

```

```
>>> import ctypes
>>> lib = ctypes.cdll.LoadLibrary("./libsample.so")
>>> print_chars = lib.print_chars
>>> print_chars.argtypes = (ctypes.c_char_p,)
>>> print_chars(b'Hello World')
48 65 6c 6c 6f 20 57 6f 72 6c 64
>>> print_chars(b'Hello\x00World')
48 65 6c 6c 6f
```

```
>>> print_chars('Hello World')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ctypes.ArgumentError: argument 1: <class 'TypeError'>: wrong
type
>>>
```

UTF-8

```
>>> print_chars('Hello World'.encode('utf-8'))
48 65 6c 6c 6f 20 57 6f 72 6c 64
>>>
```

Swig Cython

## 15.14 Unicode

### 15.14.1

Python C

### 15.14.2

C Python Unicode

## Python 如何调用 C 库函数

在 Python 中调用 C 库函数，通常使用 `char *` 和 `wchar_t *` 类型的指针，以及 `int` 类型的参数。

```
void print_chars(char *s, int len) {
    int n = 0;
    while (n < len) {
        printf("%2x ", (unsigned char) s[n]);
        n++;
    }
    printf("\n");
}

void print_wchars(wchar_t *s, int len) {
    int n = 0;
    while (n < len) {
        printf("%x ", s[n]);
        n++;
    }
    printf("\n");
}
```

在 Python 中调用 `print_chars()` 函数，通常使用 `UTF-8` 编码。

```
static PyObject *py_print_chars(PyObject *self, PyObject
*args) {
    char *s;
    Py_ssize_t len;

    if (!PyArg_ParseTuple(args, "s#", &s, &len)) {
        return NULL;
    }
}
```



```
print_chars(s, len);
Py_RETURN_NONE;
}
```

wchar\_t  
wchar\_t

```
static PyObject *py_print_wchars(PyObject *self, PyObject
*args) {
    wchar_t *s;
    Py_ssize_t len;

    if (!PyArg_ParseTuple(args, "u#", &s, &len)) {
        return NULL;
    }
    print_wchars(s, len);
    Py_RETURN_NONE;
}
```

Unicode code point value

```
>>> s = 'Spicy Jalape\u00f1o'
>>> print_chars(s)
53 70 69 63 79 20 4a 61 6c 61 70 65 c3 b1 6f
>>> print_wchars(s)
53 70 69 63 79 20 4a 61 6c 61 70 65 f1 6f
>>>
```

print\_chars() UTF-8  
print\_wchars() Unicode  
Unicode code point value

## 15.14.3 `print`

이 코드는 `print` 함수가 문자열 객체를 출력할 때 사용하는 `print_chars` 함수를 정의하고 있다. 이 함수는 `PyObject` 타입의 `self`와 `args`를 인자로 받는다. `args`는 `PyObject` 타입의 객체로, `PyArg_ParseTuple` 함수를 사용하여 `char *`와 `Py_ssize_t` 타입의 변수를 추출한다. 이 함수는 `print_chars` 함수를 호출하고, `Py_RETURN_NONE`를 반환한다.

```
static PyObject *py_print_chars(PyObject *self, PyObject
*args) {
    char *s;
    Py_ssize_t len;

    /* accepts bytes, bytearray, or other byte-like object */
    if (!PyArg_ParseTuple(args, "y#", &s, &len)) {
        return NULL;
    }
    print_chars(s, len);
    Py_RETURN_NONE;
}
```

이 코드는 `print` 함수가 유니코드 문자열 객체를 출력할 때 사용하는 `print_unicode` 함수를 정의하고 있다. 이 함수는 `PyObject` 타입의 `self`와 `args`를 인자로 받는다. `args`는 `PyObject` 타입의 객체로, `PyArg_ParseTuple` 함수를 사용하여 `char *`와 `wchar_t *` 타입의 변수를 추출한다. 이 함수는 `print_unicode` 함수를 호출하고, `Py_RETURN_NONE`를 반환한다. <http://www.python.org/dev/peps/pep-0393>는 유니코드 문자열 객체를 출력하는 방법에 대한 자세한 정보를 제공한다.

이 코드는 `print` 함수가 `UnicodeError` 예외를 처리하는 방법을 보여준다. 이 함수는 `UnicodeError` 예외를 발생시키고, `UnicodeError` 예외를 처리하는 방법을 보여준다. 이 함수는 `UnicodeError` 예외를 발생시키고, `UnicodeError` 예외를 처리하는 방법을 보여준다.

```
>>> import sys
>>> s = 'Spicy Jalape\u00f1o'
>>> sys.getsizeof(s)
87
>>> print_chars(s)
53 70 69 63 79 20 4a 61 6c 61 70 65 c3 b1 6f
>>> sys.getsizeof(s)
103
>>> print_wchars(s)
53 70 69 63 79 20 4a 61 6c 61 70 65 f1 6f
>>> sys.getsizeof(s)
163
>>>
```

```
static PyObject *py_print_chars(PyObject *self, PyObject
*args) {
    PyObject *obj, *bytes;
    char *s;
    Py_ssize_t len;

    if (!PyArg_ParseTuple(args, "U", &obj)) {
        return NULL;
    }
    bytes = PyUnicode_AsUTF8String(obj);
    PyBytes_AsStringAndSize(bytes, &s, &len);
    print_chars(s, len);
    Py_DECREF(bytes);
    Py_RETURN_NONE;
}
```

```

    wchar_t Python
    ASCII

```

U+0000~U+FFFF  
wchar\_t \*  
wchar\_t PyArg\_ParseTuple(  
) "u#"

Unicode  
C

```
static PyObject *py_print_wchars(PyObject *self, PyObject
*args) {
    PyObject *obj;
    wchar_t *s;
    Py_ssize_t len;

    if (!PyArg_ParseTuple(args, "U", &obj)) {
        return NULL;
    }
    if ((s = PyUnicode_AsWideCharString(obj, &len)) == NULL) {
        return NULL;
    }
    print_wchars(s, len);
    PyMem_Free(s);
    Py_RETURN_NONE;
}
```

PyUnicode\_AsWideCharString() wchar\_t  
C  
bug

Python

[bugs.python.org/issue16254](https://bugs.python.org/issue16254)

Python C UTF-8  
Python

```
static PyObject *py_print_chars(PyObject *self, PyObject
*args) {
    char *s = 0;
    int len;
    if (!PyArg_ParseTuple(args, "es#", "encoding-name", &s,
&len)) {
        return NULL;
    }
    print_chars(s, len);
    PyMem_Free(s);
    Py_RETURN_NONE;
}
```

Unicode

```
static PyObject *py_print_wchars(PyObject *self, PyObject
*args) {
    PyObject *obj;
    int n, len;
    int kind;
    void *data;

    if (!PyArg_ParseTuple(args, "U", &obj)) {
        return NULL;
    }
    if (PyUnicode_READY(obj) < 0) {
        return NULL;
    }
}
```

```

len = PyUnicode_GET_LENGTH(obj);
kind = PyUnicode_KIND(obj);
data = PyUnicode_DATA(obj);

for (n = 0; n < len; n++) {
    Py_UCS4 ch = PyUnicode_READ(kind, data, n);
    printf("%x ", ch);
}
printf("\n");
Py_RETURN_NONE;
}

```

PyUnicode\_KIND() PyUnicode\_DATA() Unicode PEP 393 kind 8 16 32 data PyUnicode\_READ()

Python Unicode C UTF-8 UTF-8 UTF-8 Unicode <https://docs.python.org/3/c-api/Unicode.html>

## 15.15 C Python

### 15.15.1

## 15.15.1 C Strings and Python Unicode Objects

### 15.15.2 Py\_BuildValue()

Py\_BuildValue() takes a C string and its length and returns a Python Unicode object.

```
char *s;      /* Pointer to C string data */
int len;      /* Length of data */

/* Make a bytes object */
PyObject *obj = Py_BuildValue("y#", s, len);
```

Py\_BuildValue() can also be used to create a Python Unicode object from a C string and its length, using the "s" format code.

```
PyObject *obj = Py_BuildValue("s#", s, len);
```

PyUnicode\_Decode() takes a C string and its length and returns a Python Unicode object.

```
PyObject *obj = PyUnicode_Decode(s, len, "encoding",
    "errors");

/* Examples */
obj = PyUnicode_Decode(s, len, "latin-1", "strict");
obj = PyUnicode_Decode(s, len, "ascii", "ignore");
```

wchar\_t \*, len  
Py\_BuildValue()

```
wchar_t *w;    /* Wide character string */
int len;       /* Length */

PyObject *obj = Py_BuildValue("u#", w, len);
```

PyUnicode\_FromWideChar()

```
PyObject *obj = PyUnicode_FromWideChar(w, len);
```

Unicode Python

### 15.15.3

C Python I/O C ASCII Latin-1 UTF-8

Python C NULL



## 15.16 字符串和C语言

### 15.16.1 字符串

字符串在C语言中是以字符数组的形式存在的。C语言中的字符串是以空字符'\0'结尾的。Python中的字符串是以Unicode编码的。Python中的字符串可以包含任何Unicode字符，包括UTF-8、UTF-16、UTF-32等。Python中的字符串是不可变的，而C语言中的字符串是可变的。

### 15.16.2 字符串操作

字符串操作函数在C语言中是非常多的，这里我们只介绍几个常用的。

```
/* Some dubious string data (malformed UTF-8) */

const char
*sdata = "Spicy Jalape\x3\xbf
o\xae
";
int
slen = 16;

/* Output character data */

void
```

```

print_chars(char
*s, int
len) {
    int
n = 0;
    while
(n < len) {
        printf("%2x ", (unsigned char
) s[n]);
        n++;
    }
    printf("\n
");
}

```

1. 在 C 语言中，字符串是以空字符 '\0' 结尾的字符数组。在 Python 中，字符串是不可变的序列。为了在 C 语言中处理 Python 字符串，我们需要将 Python 字符串转换为 C 字符串。这可以通过使用 `print_chars` 函数来实现，该函数将 C 字符串打印到标准输出。

2. 在 Python 中，字符串是不可变的。这意味着一旦创建，字符串就不能被修改。在 C 语言中，字符串是可变的，因为它们存储在内存中的连续位置，并且可以通过指针进行修改。为了在 Python 中处理 C 字符串，我们需要将 C 字符串转换为 Python 字符串。这可以通过使用 `print_chars` 函数来实现，该函数将 C 字符串打印到标准输出。

```

/* Return the C string back to Python */

static

```

```

PyObject *py_retstr(PyObject *self, PyObject *args) {
    if
    (!PyArg_ParseTuple(args, "")) {
        return
    NULL;
    }
    return
    PyUnicode_Decode(sdata, slen, "utf-8", "surrogateescape");
}

/* Wrapper for the print_chars() function */

static
PyObject *py_print_chars(PyObject *self, PyObject *args) {
    PyObject *obj, *bytes;
    char
    *s = 0;
    Py_ssize_t len;

    if
    (!PyArg_ParseTuple(args, "U", &obj)) {
        return
    NULL;
    }

    if
    ((bytes = PyUnicode_AsEncodedString(obj, "utf-8", "surrogateescape"))
    == NULL) {
        return
    NULL;
    }
    PyBytes_AsStringAndSize(bytes, &s, &len);
    print_chars(s, len);
    Py_DECREF(bytes);
}

```

```
Py_RETURN_NONE;
}
```

```
>>> s = retstr()
>>> s
'Spicy Jalapeño\udcaae'
>>> print_chars(s)
53 70 69 63 79 20 4a 61 6c 61 70 65 c3 b1 6f ae
>>>
    PythonC
C
```

C  
 Unicode  
 Python  
 Python  
 C  
 Python

```

    Unicode
error policystrict
ignorereplace

```

```
>>> raw = b'Spicy Jalape\x3c3\x3b1
```

```
o\x3c3ae
```

```
,
```

```
>>> raw.decode('utf-8','ignore')
```

```
'Spicy Jalapeño'
```

```
>>> raw.decode('utf-8','replace')
```

```
'Spicy Jalapeño?'
```

```
>>>
```

surrogateescape low-half  
 \udcXX XX

```
>>> raw.decode('utf-8','surrogateescape')
```

```
'Spicy Jalapeño\udca3'
```

```
>>>
```

\udca3  
Unicode  
 surrogateescape

```
>>> s = raw.decode('utf-8', 'surrogateescape')
```

```
>>> <strong>print</strong>(s)
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
UnicodeEncodeError: 'utf-8' codec can't encode character
```

```
'\udca3'
```

```
in position 14: surrogates not allowed
```

```
>>>
```

Python C surrogateescape

```
>>> s
'Spicy Jalapeño\u201c\u201e'
>>> s.encode('utf-8','surrogateescape')
b'Spicy Jalape\u201c\u201e\u201c\u201e'
>>>
```

Python

Python os.listdir() 5.15

## PEP 383

<http://www.python.org/dev/peps/pep-0383>

## 15.17 □□□□□□C□□□□

## 15.17.1 ☐ ☐

**C**

## 15.17.2 ☐☐☐☐

[illegible]

```
static
PyObject *py_get_filename(PyObject *self, PyObject *args) {
    PyObject *bytes;
    char
    *filename;
    Py_ssize_t len;
    if
    (!PyArg_ParseTuple(args, "O&", PyUnicode_FSConverter, &bytes))
    {
        return
        NULL;
    }
    PyBytes_AsStringAndSize(bytes, &filename, &len);
    /* Use filename */

    ...

    /* Cleanup and return */
}
```

```
Py_DECREF(bytes)
Py_RETURN_NONE;
}
```

```
PyObject *PyObject_Multiply(
    PyObject *a, PyObject *b) {
```

```
PyObject *obj;    /* Object with the filename */

PyObject *bytes;
char
*filename;
Py_ssize_t len;

bytes = PyUnicode_EncodeFSDefault(obj);
PyBytes_AsStringAndSize(bytes, &filename, &len);
/* Use filename */

...
/* Cleanup */

Py_DECREF(bytes);
```

```
PyObject *bytes;  
char
```

```
*filename;  
Py_ssize_t len;
```

```
bytes = PyUnicode_EncodeFSDefault(obj);
PyBytes_AsStringAndSize(bytes, &filename, &len);
/* Use filename */
```

```
...
/* Cleanup */
```

```
Py_DECREF(bytes);
```

# Python

```
/* Turn a filename into a Python object */
```

**char**

char



```

*filename;      /* Already set */

int
filename_len; /* Already set */

PyObject *obj = PyUnicode_DecodeFSDefaultAndSize(filename,
filename_len);

```

## 15.17.3 文件

Python 文件操作与 C 语言文件操作类似，但 Python 文件操作更简单。Python 文件操作与 C 语言文件操作的主要区别在于，Python 文件操作不需要打开文件，而 C 语言文件操作需要打开文件。

## 15.18 字符串与 C 语言

### 15.18.1 字符串

Python 字符串与 C 语言字符串类似，但 Python 字符串更简单。Python 字符串与 C 语言字符串的主要区别在于，Python 字符串不需要转义字符，而 C 语言字符串需要转义字符。

### 15.18.2 字符串

## PyObject\_AsFileDescriptor() 関数

```
PyObject *fobj;          /* File object (already obtained
somehow) */

int
fd = PyObject_AsFileDescriptor(fobj);
if
(fd < 0) {
    return
    NULL;
}
```

PyObject\_AsFileDescriptor() は、PyObject\* fobj を取り、fd を返す。fd は、fdopen() で開かれたファイルディスクリプタの fd を渡す。fd は、fdopen() で開かれたファイルディスクリプタの fd を渡す。fd は、fdopen() で開かれたファイルディスクリプタの fd を渡す。

PyObject\_AsFileDescriptor() は、PyObject\* fobj を取り、fd を返す。fd は、fdopen() で開かれたファイルディスクリプタの fd を渡す。fd は、fdopen() で開かれたファイルディスクリプタの fd を渡す。fd は、fdopen() で開かれたファイルディスクリプタの fd を渡す。

PyObject\_AsFileDescriptor() は、PyObject\* fobj を取り、fd を返す。fd は、fdopen() で開かれたファイルディスクリプタの fd を渡す。fd は、fdopen() で開かれたファイルディスクリプタの fd を渡す。fd は、fdopen() で開かれたファイルディスクリプタの fd を渡す。

```
int
fd;          /* Existing file descriptor (already open) */
```

```
PyObject *fobj = PyFile_FromFd(fd,
"filename", "r", -1, NULL, NULL, NULL, 1);
```

PyFile\_FromFd() 调用 open() 函数，  
mode 为 "r"，encoding 为 NULL，errors 为  
newline 为 ""。

### 15.18.3 文件操作

Python 的 C 扩展模块中，  
Python 的 io 模块提供了 I/O 操作。  
C 扩展模块中，I/O 操作  
通常通过 Python 的 io 模块实现。

ownership 属性，  
C 扩展模块 Python 的  
C 扩展模块中，  
Python 的 PyFile\_FromFd() 函数  
1 参数，  
Python 的

C 的 I/O 操作，  
fdopen() 函数，  
FILE \* 参数，  
I/O 操作，  
Python 的  
C 的 stdio 函数，  
fclose() 函数。

Python 调用 C 函数，需要包含头文件 `<stdio.h>`，并声明 `FILE` 指针。

## 15.19 C 函数调用

### 15.19.1 读取文件

Python 调用 C 函数，需要包含头文件 `<stdio.h>`，并声明 `FILE` 指针。

### 15.19.2 写入文件

Python 调用 C 函数，需要包含头文件 `<stdio.h>`，并声明 `FILE` 指针。

Python 调用 C 函数，需要包含头文件 `<stdio.h>`，并声明 `FILE` 指针。

```
#define CHUNK_SIZE 8192

/* Consume a "file-like" object and write bytes to stdout */

static
PyObject *py_consume_file(PyObject *self, PyObject *args) {
    PyObject *obj;
    PyObject *read_meth;
    PyObject *result = NULL;
```

```

    PyObject *read_args;

    if
(!PyArg_ParseTuple(args, "0", &obj)) {
        return
NULL;
    }

    /* Get the read method of the passed object */

    if
((read_meth = PyObject_GetAttrString(obj, "read")) == NULL) {
        return
NULL;
    }

    /* Build the argument list to read() */

    read_args = Py_BuildValue("(i)", CHUNK_SIZE);
    while
(1) {
        PyObject *data;
        PyObject *enc_data;
        char
*buf;
        Py_ssize_t len;

        /* Call read() */

        if
((data = PyObject_Call(read_meth, read_args, NULL)) == NULL) {
            goto
final;
        }

```

```

    /* Check for EOF */

    if
(PySequence_Length(data) == 0) {
        Py_DECREF(data);
        break
;
    }

    /* Encode Unicode as Bytes for C */

    if
((enc_data=PyUnicode_AsEncodedString(data,"utf-
8","strict"))==NULL) {
        Py_DECREF(data);
        goto
final;
    }

    /* Extract underlying buffer data */

    PyBytes_AsStringAndSize(enc_data, &buf, &len);

    /* Write to stdout (replace with something more useful) */

    write(1, buf, len);

    /* Cleanup */

    Py_DECREF(enc_data);
    Py_DECREF(data);
}
result = Py_BuildValue("");

final:

```

```

    /* Cleanup */

    Py_DECREF(read_meth);
    Py_DECREF(read_args);
    return

result;
}

```

StringIO

```

>>> import io

>>> f = io.StringIO('Hello\n
World\n
')
>>> import sample

>>> sample.consume_file(f)
Hello
World
>>>

```

## 15.19.3

C

## Python C API 関数呼び出し Python 関数呼び出し

read() 関数呼び出し  
PyObject\_Call() 関数呼び出し  
EOF 関数呼び出し  
PySequence\_Length() 関数呼び出し  
0

I/O 関数呼び出し  
Unicode 関数呼び出し  
C 関数呼び出し  
関数呼び出し

```
... /* Call read() */

if
((data = PyObject_Call(read_meth, read_args, NULL)) == NULL) {
    goto
final;
}

/* Check for EOF */

if
(PySequence_Length(data) == 0) {
    Py_DECREF(data);
    break
;
}
if
```



```

(!PyBytes_Check(data)) {
    Py_DECREF(data);
    PyErr_SetString(PyExc_IOError, "File must be in binary
mode");
    goto

final;
}

/* Extract underlying buffer data */

PyBytes_AsStringAndSize(data, &buf, &len);
...

```

PyObject \*  
 Py\_DECREF()

write()  
 Python Unicode

readline() read\_into()  
 read() write() C

## 15.20 C

## 15.20.1

        C          Python

## 15.20.2

        C          

```
static
PyObject *py_consume_iterable(PyObject *self, PyObject *args)
{
    PyObject *obj;
    PyObject *iter;
    PyObject *item;

    if
(!PyArg_ParseTuple(args, "O", &obj)) {
        return
NULL;
    }
    if
((iter = PyObject_GetIter(obj)) == NULL) {
        return
NULL;
    }
    while
((item = PyIter_Next(iter)) != NULL) {
        /* Use item */
    }
}
```

```

        ...
        Py_DECREF(item);
    }
    Py_DECREF(iter);
    return
Py_BuildValue("");
}

```

## 15.20.3 遍历

Python 遍历 Python 的 `PyObject_GetIter()` 返回一个 Python 的 `iter()` 对象，这个对象实现了 `PyIter_Next()` 方法，返回下一个元素，如果没有下一个元素，返回 `NULL`。遍历结束后，需要调用 `Py_DECREF()` 来释放内存。

## 15.21 异常处理

### 15.21.1 异常

Python 异常处理使用 `try` 和 `except` 语句。Python 的 `try` 语句可以捕获异常，并执行相应的处理。Python 的 `except` 语句可以捕获异常，并执行相应的处理。Python 的 `try` 语句可以捕获异常，并执行相应的处理。

### 15.21.2 异常处理



```
File "example.py", line 19 in
```

```
<module>  
Segmentation fault
```

Python CPython  
Python

### 15.21.3

Python  
Python  
Python  
Python  
Python

Python  
Python  
Python  
Python  
Python

Python  
Python  
Python  
Python  
Python

## Appendix A

This appendix contains information about Python 3. The information in this appendix is intended to help you understand the differences between Python 2 and Python 3. The information in this appendix is intended to help you understand the differences between Python 2 and Python 3.

This appendix contains information about Python 3. The information in this appendix is intended to help you understand the differences between Python 2 and Python 3. The information in this appendix is intended to help you understand the differences between Python 2 and Python 3.

### A.1

<http://docs.python.org>

This appendix contains information about Python 3. The information in this appendix is intended to help you understand the differences between Python 2 and Python 3. The information in this appendix is intended to help you understand the differences between Python 2 and Python 3.

<http://www.python.org/dev/peps>

This appendix contains information about Python 3. The information in this appendix is intended to help you understand the differences between Python 2 and Python 3. The information in this appendix is intended to help you understand the differences between Python 2 and Python 3.

<http://pyvideo.org>

PyCon  
Python  
Python  
Python 3

<http://code.activestate.com/recipes/langs/python>

ActiveState  
Python  
300  
Python 3  
Python 3

<http://stackoverflow.com/questions/tagged/python>

Stack Overflow  
175000  
Python  
5000  
Python  
3  
Python 3

## A.2 Python

Python Python 3

- *Learning Python* Mark Lutz  
O'Reilly&Associates 2009
- *The Quick Python Book*  
Vernon Ceder Manning 2010
- *Python Programming for the Absolute Beginner* Michael Dawson  
Course Technology PTR 2010
- *Beginning Python: From Novice to Professional* Magnus Lie Hetland  
Apress 2008
- *Programming in Python 3* Mark Summerfield  
Addison-Wesley 2010

## A.3

Python 3



- *Programming Python* □□□□□□□ Mark Lutz □ O'Reilly & Associates □□□ 2010 □□
- *Python Essential Reference* □□□□□□□ David Beazley □ Addison-Wesley □□ □ 2009 □□
- *Core Python Applications Programming* □□□□□□□ Wesley Chun □ Prentice Hall □□□ 2012 □□
- *The Python Standard Library by Example* □□□ Doug Hellmann □ Addison-Wesley □□□ 2011 □□
- *Python 3 Object Oriented Programming* □□□ Dusty Phillips □ Packt Publishing □□□ 2010 □□
- *Porting to Python 3* □□□ Lennart Regebro □ CreateSpace □□□ 2011 □□ <http://python3porting.com> □





□□□□□□□□

□□□□□□□

□□□□([www.epubit.com.cn](http://www.epubit.com.cn))□□□□□□□□□□  
IT□□□□□□□□□□2015□8□□□□□□

□□□□□□□□□□□□□□20□□□IT□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□POD□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□

异步社区
技术圈 · 图书 电子书 文章

2
写作

# 我们一岁啦

## 异步社区成立一周年大型活动开启

周年庆满减促销 | 满100元减20元、满150元减35元、满200元减50元

CCIE路由和交换认证考试指南 (第5版) (第1卷)

数据科学实战手册 (R+Python)

软技能: 代码之外的生存指南

Python密码学编程

Python游戏编程快速上手

机器学习项目开发实战

树莓派Python编程入门与实战 (第2版)

像计算机科学家一样思考Python (第2版)

### 近期活动

异步社区成立一周年大型赠书活动开启!

异步社区的来历 异步社区是人民邮电出版社旗下IT专业, 业图书旗舰社区, 于2015年8月上线运营。异步社区依托于人民邮电出版社20余年的IT专业...

**猫叔郭志敬** 2016-08-02  
阅读 575 推荐 2 收藏 0 评论 8

**2016 iWeb峰会北京站即将开启, 为HTML5呐喊!**

每一次振臂高呼顿时行业的影响, 每一天无数人兢兢业业的勤奋, 2016雄起! 来吧, 8月27日, HTML5峰会北京站, 我在这里, 等你来, 为HTML5呐喊!...

**猫叔郭志敬** 2016-07-29  
阅读 60 推荐 1 收藏 0 评论 0

### 每周半价电子书

**树莓派Python编程入门与实战 (第2版)**

[美] Richard Blum 勃鲁姆, Christine Bresnahan 布莱斯纳罕 (作者) 陈晓明 马立新 (译者)

□□□□□□□□

□□□□

□□□□□□□□□□IT□□□□□□□□Web□□□□□□□□  
 □□□□□□□□□□□□□□□□□□□□1000□□□□□□400□  
 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□

□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□

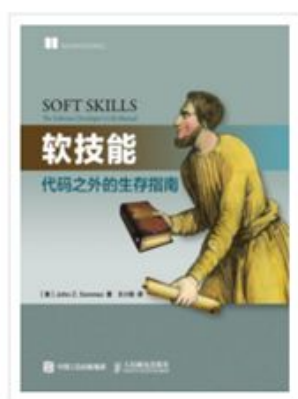
□□□□

□□□□□□□□□□□□□□□□100□□=1□□□□□□□□

□   □□□□□□□□□□□□□□□□□□□□□□

□□□□

“57AWG” “ ”8



## 软技能：代码之外的生存指南

[美]约翰 Z. 森梅兹 (John Z. Sonmez) (作者)

王小刚 (译者)

杨海玲 (责任编辑)



分享

6

推荐



想读

9.0K

阅读

这是一本真正从“人”（而非技术也非管理）的角度关注软件开发人员自身发展的书。书中论述的内容既涉及生活习惯，又包括思维方式，凸显技术中“人”的因素，全面讲解软件行业从业人员所需知道的所有“软技能”。

本书聚焦于软件开发人员生活的方方面面，从揭秘面试的流程到精耕细作出一份杀手级简历，从创建大受欢迎的博客到打造你的个人品牌，从提高自己工作效率到如何与“拖延症”做斗争，甚至包括如何投资不动产，如何关注自己的健康。

本书共分为职业篇、自我营销篇、学习篇、生产力篇、理财篇、健身篇、精神篇等七篇，概括了软件行业从业人员所需的“软技能”。

● 纸质版 ~~¥59.00~~ ¥46.02 (7.8折)

● 电子版 ¥35.00

● 电子版 + 纸质版 ¥59.00

现在购买

下载PDF样章

配套文件下载

100

□□

□□□□□□Markdown□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□

□□□□□IT□□□□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□□□□□





0000



00000



00000



□□□□



QQ□□368449889

□□□□□ [www.epubit.com.cn](http://www.epubit.com.cn)

□□□□□ □□□□

□□□□□ @□□□□□□□@□□□□□□□-□□□□□□□

□□&□□□ contact@epubit.com.cn